le cnam

ÉCOLE DOCTORALE INFORMATIQUE, TÉLÉCOMMUNICATION ET ÉLECTRONIQUE (PARIS)

ÉQUIPES ISID/VERTIGO – LABORATOIRE CEDRIC

THÈSE présentée par :

Zeinab HMEDEH

soutenue le : **10 décembre 2013**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline/ Spécialité : Informatique

Indexation pour la recherche par le contenu textuel de flux RSS

THÈSE dirigée par :

M DU MOUZA Cédric	MCF-HDR, Cnam
M TRAVERS Nicolas	MCF, Cnam

RAPPORTEURS :

Mme COLLET Christine	PR, Institut Polytechnique de Grenoble (Grenoble INP)
M GROSS-AMBLARD David	PR, Université de Rennes 1

JURY :

Mme COLLET Christine	PR, Institut Polytechnique de Grenoble (Grenoble INP)
M GROSS-AMBLARD David	PR, Université de Rennes 1
M AMANN Brend	PR, Université Pierre et Marie Curie
M LAURENT Dominique	PR, Université de Cergy-Pontoise
M DU MOUZA Cédric	MCF-HDR, Cnam
M TRAVERS Nicolas	MCF, Cnam

Aux deux personnes qui nous ont quitté trop tôt :

À la femme qui m'a tout offert et j'ai rien pu lui offrir, à ma mère,

أمي... كلما حققت نجاحًا أو تفوقًا... تذكرتك
أتذكرك في فرحي كما في حزني ... في نجاحي كما في فشلي
فأنت أول و آخر إنسانة رحلت و بكيت فراقها ألمًا و حسرة

Et à Michel . . .

Commentaire

Cette thèse a été dirigée par Michel SCHOLL [1] jusqu'à son départ en Novembre 2011 et a été financée par une bourse doctorale de trois ans par le CNRS-Liban [2].

1. http ://cedric.cnam.fr/∼scholl/hommage
2. http ://www.cnrs.edu.lb

Remerciements

Mes remerciements s'adressent plus particulièrement à Michel, je le remercie pour sa confiance de m'avoir acceptée en stage de master puis en thèse. Je le remercie également d'avoir dirigé ma thèse et travaillé avec moi jusqu'à la dernière minute de sa vie.

Je suis très reconnaissante à mes encadrants Cédric et Nicolas, c'est grâce à leur encadrement et leur orientation que j'ai pu réaliser ce travail. Je suis plus que chanceuse de les avoir eu comme encadrants, j'ai été gâtée par leur gentillesse et leur patience.

Je tiens à remercier Madame et Messieurs les membres du jury de m'avoir fait l'honneur de juger mon travail de thèse. Je remercie sincèrement Madame Christine COLLET et Monsieur David GROSS-AMBLARD d'avoir porté autant d'intérêt à ce travail en acceptant d'être les rapporteurs de ma thèse. Je les remercie pour leurs remarques qui ont permis d'améliorer ce manuscrit. Je remercie également Monsieur Bernd AMANN et Monsieur Dominique LAURENT d'avoir participé à mon jury de thèse et au jury de mon évaluation à mi-parcours.

Je remercie Monsieur Vassilis CHRISTOPHIDES pour la collaboration pendant ma thèse pour son aide et ses conseils, je suis ravie d'avoir travaillé avec lui. Un merci à Nelly et Harry pour les différents échanges pendant cette collaboration.

Je remercie le Conseil d'Administration du CNRS libanais d'avoir financé ma thèse, en particulier, Monsieur Charles TABET pour sa gentillesse.

J'exprime ma gratitude à mes enseignants à l'université libanaise Monsieur Ali AWADA et Monsieur Bilal CHEBARO pour leurs encouragements et leurs soutiens dès mes premiers pas dans le domaine de l'informatique.

Ensuite, mes remerciements s'adressent à tous les membres des équipes ISID et Vertigo pour leur soutien pendant ma thèse. Particulièrement, je remercie mes amis les doctorants avec qui j'ai partagé les plus durs et les plus beaux moments de ma thèse : Amina, Anh, Feten, Houda, Nabil, Ryadh et Sarah ; et les ingénieurs Pascal et Rodney. Un très grand merci est à Lydia qui était à côté de moi dès mon premier jour au cnam, je la remercie pour sa gentillesse et son amitié. Je n'oublie pas les stagiares de la promotion 2010 du master SID avec qui j'ai passé des moments inoubliables : Ryme, Yassmine et Hind.

Je remercie toutes mes amies qui j'ai eu la chance de connaître : Amany Sabra, Khatleen,

Ronza, Darine, Amani Chamis et Sally.

Je remercie également mes amis de BD Damien et Imen avec qui j'ai passé de bonnes journées à BDA. Un merci à Imen pour son amitié et son soutien durant la période la plus difficile de ma thèse qu'on a partagée ensemble.

Mes plus grands remerciements vont à mon père pour son encouragement et son soutien pendant ma thèse. Je remercie du fond du cœur ma sœur Mariam de m'avoir supportée surtout les derniers mois de ma thèse. Elle était toujours là pour me remonter le moral. Un merci très particulier à ma cousine Ola qui m'a beaucoup encouragée et soutenue. Un très très Grand Grand Merci à mes frères Hassan et Ali pour leurs sacrifices.

Résumé

Afin de réduire l'intervalle de temps nécessaire entre la publication de l'information sur le Web et sa consultation par les utilisateurs, les sites Web reposent sur le principe de la Syndication Web. Les fournisseurs d'information diffusent les nouvelles informations à travers des flux RSS auxquels les utilisateurs intéressés peuvent s'abonner. L'objectif de la thèse est de proposer un système de notification passant à l'échelle du Web, prenant en considération un grand nombre d'utilisateurs et un débit élevé d'items. Nous proposons un index basé sur les mots-clés des requêtes utilisateurs permettant de retrouver ceux-ci dans les items des flux. Trois structures d'indexation de souscriptions sont présentées. Un modèle analytique pour estimer le temps de traitement et l'espace mémoire de chaque structure est détaillé. Nous menons une étude expérimentale approfondie de l'impact de plusieurs paramètres sur ces structures. Pour les souscriptions jamais notifiées, nous adaptons les index étudiés pour prendre en considération leur satisfaction partielle. Afin de réduire le nombre d'items reçus par l'utilisateur, nous intégrons une deuxième phase de filtrage par nouveauté et diversité considérant l'ensemble d'items déjà reçus par l'utilisateur.

Mots-clés : Syndication Web, Flux RSS, Système Publication/Souscription, Indexation, Diversité, Nouveauté.

Abstract

Based on a Publish/Subscribe paradigm, Web Syndication formats such as RSS have emerged as a popular means for timely delivery of frequently updated Web content. According to these formats, information publishers provide brief summaries of the content they deliver on the Web, while information consumers subscribe to a number of RSS feeds and get informed about newly published items. The goal of this thesis is to propose a notification system which scales on the Web. To deal with this issue, we should take into account the large number of users on the Web and the high publication rate of items. We propose a keyword-based index for user subscriptions to match it on the fly with incoming items. We study three indexing techniques for user subscriptions. We present analytical models to estimate memory requirements and matching time. We also conduct a thorough experimental evaluation to exhibit the impact of critical workload parameters on these structures. For the subscriptions which are never notified, we adapt the indexes to support a partial matching between subscriptions and items. We integrate a diversity and novelty filtering step in our system in order to decrease the number of notified items for short subscriptions. This filtering is based on the information delivered in the set of items already received by the user.

Key-words : Web Syndication, RSS Feed, Publish/Subscribe system, Indexing, Diversity, Novelty.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte

La quantité d'informations publiée sur le Web connaît une croissance exponentielle depuis plusieurs années. De ce fait, suivre les informations qui y circulent devient un véritable défi aussi bien pour rechercher les données pertinentes que pour les traiter rapidement. Un utilisateur qui souhaite être notifié sur un sujet particulier doit se contenter de simples agrégateurs dédiés qui n'ont pas une grande capacité d'expressivité. En effet, ceux-ci sont confrontés à une énorme quantité d'informations et un grand nombre d'utilisateurs. La difficulté pour ces agrégateurs est double avec un passage à l'échelle en quantité de données (plusieurs milliers d'événements à la minute) et en quantité de requêtes (des dizaines de millions de requêtes à maintenir), tout en garantissant des temps de réponses très courts à chacune de ces requêtes.

La syndication Web est une application particulière de la publication/souscription. Ce modèle repose principalement sur deux formats de données XML (eXtensible Markup Language) : RSS et Atom. RSS (*Really Simple Syndication*) permet de diffuser de manière simple et hiérarchisée les informations. Atom est un autre format standard de la syndication Web normalisé par l'IETF [1]. Bien que les deux formats Atom et RSS aient quelques différences de structures, ils possèdent les mêmes fonctionnalités, et beaucoup de gens utilisent, par abus de langage, le terme « RSS » pour désigner indifféremment RSS ou bien Atom. De nombreux sites Web et blogs proposent la diffusion de leurs actualités sous forme de flux RSS. Ce standard permet à un site Web de diffuser largement ses actualités tout en récupérant un grand nombre de visiteurs grâce aux liens hypertextes permettant au lecteur de lire la suite de l'actualité en ligne sur ce site. Un fichier RSS est un simple fichier texte au format XML présentant les publications d'informations de manière organisée en items. Généralement, il contient un ensemble de méta-données tel que : le titre de l'item, une courte description, la date de sa dernière publication et le

1. Internet Engineering Task Force

lien du site Web. Un utilisateur intéressé par les changements d'informations sur un site Web, va, au lieu de retourner régulièrement sur celui-ci, s'abonner à son flux RSS.

Un « Agrégateur RSS » permettra de collecter les nouvelles informations provenant des flux RSS et de les notifier à l'utilisateur. L'agrégateur est un logiciel installé localement (*par ex.*, iTunes, Opera...) ou bien un site Web en ligne (*par ex.*, MSN, Yahoo ...). Il suit plusieurs flux de données en visitant les différents sites Web pour récupérer les nouvelles informations publiées et les présente à l'utilisateur. De plus, il permet de catégoriser les informations collectées et une bonne visualisation du contenu. L'utilisateur peut personaliser l'affichage des informations collectées en précisant par exemple le nombre d'items à afficher par flux. Pour chaque item, il peut choisir de l'afficher en totalité ou bien d'afficher un résumé de son contenu. L'utilisateur pourra par suite choisir de consulter la page Web du flux ou bien simplement de consulter le contenu dans son agrégateur.

Si la syndication Web facilite la diffusion de nouvelles informations publiées sur le Web, cette approche possède toutefois quelques limitations. En effet, l'utilisateur est notamment toujours lié aux publieurs d'informations. Il doit avoir connaissance des flux auxquels il s'abonne et du type d'informations qu'ils publient. L'utilisateur s'abonne également parfois à des flux alors qu'il n'est intéressé que par un sous-ensemble des informations publiées. Ainsi, afin d'améliorer la qualité des systèmes de diffusion, nous proposons un système découplant les utilisateurs des publieurs d'information, tout en introduisant un filtrage sur l'information publiée. Le flux d'information sera l'agrégat de l'ensemble des flux intégrés. Cela permettra à l'utilisateur de préciser ses intérêts uniquement sur les informations publiées.

Dans cette thèse, nous nous intéressons particulièrement aux systèmes de Publication/Souscription « *Publish/Subscribe System* ». Le terme Publication/Souscription (Pub/Sub) correspond à l'abonnement d'un utilisateur (sous forme d'une requête) à un sujet pour être notifié lorsque ce sujet apparaît dans les données. Les données sont publiées sur le Web via une application dédiée et l'utilisateur souscrit à l'information pour être notifié lors de son apparition. Ces systèmes établissent une connexion entre les publieurs (producteurs) et les souscripteurs (consommateurs). Le rôle de ces systèmes est de notifier les utilisateurs ayant des souscriptions satisfaites par les événements publiés. Ainsi, les publieurs et les souscripteurs sont découplés ce qui facilite la gestion du système. Plusieurs modèles de systèmes Pub/Sub seront détaillés dans le premier chapitre. L'approche naïve pour notifier des souscriptions est de les évaluer périodiquement sur des données statiques. Toutefois, cette approche ne permet pas le passage l'échelle en nombre de souscriptions ni de supporter un débit élevé d'événements. En effet la quantité importante de mises à jour imposent une surcharge d'évaluation des souscriptions et exigent un temps de réponse sans délai. Notre but est donc de proposer un système efficace pour évaluer des millions de souscriptions sur des données intensivement mises à jour.

1.2 Problématique

Dans le cadre de cette thèse, notre problématique se focalise sur la notification efficace de souscriptions exprimées sous la forme d'ensembles de mots-clés soumis par les utilisateurs sur un flux d'informations. La notification d'une souscription (ou « *broad matching* » de requête) consiste à vérifier si tous les mots de la souscription indexée sont contenus dans le texte de l'item publié. Du fait du trop grand nombre de souscriptions, et d'un nombre toujours croissant de publications soumises au système, nous sommes confronté à un réel problème de passage à l'échelle. Les publications d'un flux que nous appelons *items* provenant des flux RSS et Atom contiennent en grande partie du texte. Nous nous intéresserons donc à l'élaboration de structures de données et d'algorithmes de notification efficaces pour la syndication Web en se basant sur l'approche des systèmes Pub/Sub.

Compte tenu du grand débit d'items provenant des flux de syndication Web, du nombre de souscriptions associées et du délai maximum de notifications, les méthodes standards de structures de données stockées sur disque ne sont pas envisageables. En effet, nous souhaitons éviter les accès disques répétés qui ralentiraient considérablement les temps de réponses. Ainsi, nous nous concentrerons uniquement sur des structures en mémoire centrale. Plusieurs structures d'indexation sont utilisées pour les requêtes Web, se basant principalement sur des listes inverses ou bien des arbres. Dans ces structures les documents (pages Web) sont indexés, et on effectue une recherche pour une requête Web (courte). Dans le cadre de la syndication Web c'est l'inverse, ce sont les souscriptions qui sont de petites tailles qui sont indexées et traitées à la volée avec les items publiés (plus grands). Pour cela nous devons écarter les structures classiques et concevoir une structure plus adéquate pour un traitement à la volée d'un très grand nombre de souscriptions.

Une autre difficulté liée à la taille du vocabulaire publié sur le Web et du nombre de fautes d'orthographe que peuvent contenir les souscriptions soumises par les utilisateurs (chapitre 3), considérer une satisfaction totale (*c.-à-d.*, tous les mots-clés de la souscription apparaissent dans l'item pour qu'elle soit notifiée) d'une souscription peut conduire à manquer de l'information pertinente. Nous nous intéressons aussi à la satisfaction partielle des souscriptions par les items, en relâchant la contrainte du *broad match*. Nous souhaitons ainsi notifier la souscription si l'item contient de l'information suffisamment pertinente pour la souscription. Cela peut être réalisé en testant l'inclusion des termes représentatifs de la souscription (*c.-à-d.*, les plus importants ou bien les plus discriminants). Cependant, les techniques de pondération usuelles se basent des requêtes courtes et des documents larges ce qui n'est pas le cas dans le contexte de la syndication Web. Nous devons donc proposer une stratégie de pondération plus pertinente dans ce contexte. Par ailleurs, il est toujours nécessaire de garantir un traitement à la volée des items publiés. Il faut donc adapter nos algorithmes et nos structures pour intégrer la satisfaction partielle.

Enfin, une information diffusée sur le Web peut être publiée par différentes sources de données, à l'identique ou de contenu très similaire. Pour ne pas submerger l'utilisateur par

un grand nombre d'items contenant la même information nous nous intéressons à un filtrage intelligent capable de réduire l'ensemble des items notifiés pour chaque souscription. Plus précisement, nous nous focalisons sur la diversité et la nouveauté de l'information, domaine récemment étudié dans la littérature afin d'améliorer la satisfaction de l'utilisateur. Le problème de nouveauté et de diversité a été initialement utilisé pour répondre aux requêtes ambigües soumises aux moteurs de recherche, et aux requêtes sur de larges bases de données et dans les systèmes de recommandations. Ce problème est plus souvent présenté comme un problème de requêtes *top-k* dont le but est d'envoyer à l'utilisateur les k éléments les plus divers et ne contenant pas d'information redondante. Dans le cadre des systèmes Pub/Sub, le problème doit être défini autrement pour prendre en compte le taux de publication d'items, le nombre de souscriptions, mais surtout la chronologie imposée par le flux de données. Dans ce type de systèmes, lorsqu'un nouvel item est publié il est nécessaire de vérifier sa nouveauté et sa diversité pour décider s'il doit être notifié à l'utilisateur. Mais, cette vérification doit tenir compte de ce qui a déjà été notifié, sans toutefois modifier la pertinence des notifications passées. Ainsi, nous étudions dans cette thèse un système de filtrage efficace, via nouveauté et diversité, pour un traitement à la volée des items des flux RSS.

1.3 Contributions et organisation de la thèse

Dans cette thèse, nous présentons tout d'abord une étude statistique qui caractérise le comportement des flux RSS concernant leurs activités ainsi que leur contenu textuel. Celle-ci nous a permis de mieux appréhender le comportement des flux et de concevoir un système adapté à ce contexte. Nous proposons un système de publication/souscription pour la syndication Web qui permet aux utilisateurs de filtrer les items publiés par les flux RSS selon leurs intérêts. Ce filtrage néamoins n'évite pas que l'utilisateur reçoive une énorme quantité d'informations redondante, rendant négative son expérience des systèmes Pub/Sub. Pour améliorer le système, nous ajoutons un deuxième filtrage à notre système basé sur la nouveauté et la diversité de l'information contenue dans les items publiés.

La thèse est organisée comme suit :

Chapitre 2 : État de l'art Dans le premier chapitre de cette thèse, nous nous intéressons aux différentes structures d'indexation utilisées dans des contextes de Pub/Sub ou de domaines proches ainsi que les algorithmes de notification de souscriptions associées. Les listes inverses et les arbres sont deux techniques d'indexation proposées dans la littérature pour compter explicitement (*vs* implicitement) le nombre de termes de la souscription qui sont contenus dans l'item. Nous présentons différentes variantes de ces index liées au type de données traitées (structurées ou textuelles), ainsi que des travaux comparant leurs performances dans des contextes différents du nôtre.
Un état de l'art des travaux de recherche pour le filtrage par nouveauté et diversité

est également développé. La diversité de l'information a été utilisée dans plusieurs domaines pour une meilleure satisfaction de l'utilisateur. Les algorithmes utilisés pour calculer la nouveauté et la diversité dépendent du domaine dans lequel elles sont appliqués. Il existe deux familles d'algorithmes qui sont souvent utilisées pour calculer l'ensemble des éléments les plus distants : *Greedy* (incrémental) et *Swap* (*interchange*). Toutefois, ce calcul peut être périodique ou bien basé sur une fenêtre glissante, combiné avec une approche *top-k* pour choisir l'ensemble résultat qui favorise la diversité. Dans ce chapitre, nous classons les solutions proposées selon leur contexte.

Chapitre 3 : Caractérisation des flux RSS La première contribution de cette thèse est une analyse poussée du comportement des flux. Le but est ensuite de proposer des structures d'index en intégrant les spécificités des données RSS/Atom. L'étude détaillée sur les flux RSS/Atom fait l'objet de ce chapitre. Cette étude se focalise tout d'abord sur le comportement des flux en terme de débit de publication, puis sur la structure des items, et s'achève sur le vocabulaire contenu dans les items. Connaître le taux de publication des items permettra de mieux estimer la capacité de l'index en fonction du nombre de flux en entrée pour le traitement de leurs items. Il est également nécessaire de connaître la taille des items en nombre de mots-clés afin d'évaluer le modèle et la complexité des index, et de définir précisement le type d'indexation appropriée. L'étude du vocabulaire est également essentielle pour étudier les besoins en espace et en temps des différentes structures d'index, ainsi que leur évolution au cours du temps.

Chapitre 4 : Indexation des souscriptions En s'appuyant sur les structures existantes et l'étude du chapitre 2, nous proposons dans ce chapitre nos structures d'indexation et leur algorithme d'évaluation (*matching*).

Nous adaptons tout d'abord trois structures d'indexation des mots-clés des requêtes utilisateurs. La première basée sur les listes inverses et un algorithme de comptage, stocke chaque souscription dans les listes inverses de tous les termes qui la compose. La seconde structure s'appuie sur la fréquence des mots (*Ranked Key Inverted Index*) [Yan and Garcia-Molina, 1994]. Dans cette structure des listes inverses, une souscription est indexée uniquement par le terme le moins fréquent parmi l'ensemble de ses termes. La troisième structure est arborescente (*Trie*, treillis), elle repose sur la factorisation des requêtes, ayant un préfixe commun. Un ordre total des termes du vocabulaire est également nécessaire pour cette structure.

Nous décrivons ensuite le modèle analytique de chacune des solutions abordées. Cette étude permet d'estimer l'espace mémoire et le temps de *matching* et de définir un point de comparaison entre elles. Ce modèle probabiliste prend en considération la distribution des fréquences des tailles de souscriptions ainsi que celles des occurrences des termes.

Nous adaptons ensuite les structures précédement présentées pour prendre en considération la satisfaction partielle des souscriptions. Nous associons à chaque terme d'une souscription un poids et calculons un score de pertinence entre souscriptions et items. Nous utilisons le TDV (*Term Discrimination Value*) pour la pondération

des termes de notre vocabulaire. Cette technique mesure l'importance du terme comme étant sa capacité de distinguer entre les items. Un item est alors notifié si son score de pertinence dépasse un seuil fixe.

Le chapitre se conclut sur la phase d'évaluation des performances des structures présentées. Nous prêtons une attention particulière à mettre en corrélation les résultats attendus dans le modèle analytique avec les résultats obtenus sur un ensemble de tests.

Chapitre 5 : Filtrage par nouveauté et diversité Dans le but de réduire le nombre d'items notifiés à l'utilisateur, nous proposons d'ajouter une autre phase de filtrage. Ce filtrage est basé sur la nouveauté et la diversité de l'information publiée dans les items. Dans ce chapitre, nous développons la notion de nouveauté et diversité pour les items publiés par les flux RSS. Une fenêtre glissante est adaptée pour gérer l'historique des items déjà reçus par l'utilisateur. Plusieurs optimisations en espace mémoire et temps de traitement sont proposées pour un traitement efficace des items. Nous discutons ensuite des contraintes que cela implique sur le calcul de la nouveauté et diversité.

Des expériences sont réalisées pour étudier l'impact de plusieurs paramètres comme le seuil de nouveauté, la taille de la fenêtre et la taille des souscriptions sur le taux de filtrage. Les performances du système en espace mémoire et temps de traitement sont aussi étudiées. Pour finir, une comparaison de la qualité de l'information envoyée à l'utilisateur par ce filtrage est faite avec celle envoyée par un système périodique basé sur une approche *top-k*.

Chapitre 6 : Conclusion Ce chapitre présente les conclusions sur ce travail et aborde les perspectives de recherche de cette thèse.

Chapitre 2

État de l'art

2.1 Introduction

Dans ce travail de thèse, nous nous intéressons aux systèmes de type « *Publish/Subscribe* » ou *Pub/Sub* pour des souscriptions composées de mots-clés. Le but de ces systèmes est de découpler les publieurs (producteurs) et les souscripteurs (consommateurs) afin de faciliter la gestion du système, en établissant une connexion entre les publieurs et les souscripteurs . Les souscripteurs expriment leurs intérêts au moyen de souscriptions au système qui doit chercher l'ensemble des souscriptions satisfaites par les événements publiés afin de les notifier. Pour une recherche efficace des souscriptions satisfaites, une approche classique consistant à évaluer périodiquement les items publiés sur l'ensemble des souscriptions soumises au système ne permet pas le passage à l'échelle du Web (millions des souscriptions et un débit élevé d'items). L'objectif pour la syndication Web de cette thèse est de trouver un algorithme de recherche des souscriptions efficace et une structure d'indexation compacte en mémoire. Dans notre contexte, une souscription est notifiée par les items des flux contenant chacun des mots de la souscription. Ainsi, le but de l'index est de préserver la corrélation des mots-clés de la souscription pour les retrouver dans les items. Pour des raisons de performance, nous nous focaliserons sur des structures capables de tenir en mémoire centrale. En effet, au vu du nombre de souscriptions à indexer dans le système (plusieurs millions), il est inenvisageable de faire des accès disques pour retrouver les souscriptions à notifier pour chaque item à traiter. Le problème devient d'ailleurs d'autant plus difficile pour les requêtes très fréquentes, car les mots-clés populaires deviennent redondants dans le système. Ces mots apparaîtront plus dans les items et seront les plus testés pendant le traitement des items. Ainsi, nous nous efforcerons de garder cette structure en mémoire.

Ensuite, nous nous étudierons le filtrage par nouveauté et diversité des items publiés par les flux RSS, pour permettre à l'utilisateur de recevoir l'information qui l'intéresse sans être gêné par un grand nombre d'items. Le but du filtrage par nouveauté est de supprimer

l'information redondante dans les items publiés, alors que le but du filtrage par diversité est d'envoyer un ensemble minimal d'items à l'utilisateur qui couvre toutes les actualités. Ce filtrage est basé sur l'ensemble de l'historique des items notifiés à l'utilisateur. Pour les mêmes contraintes de passage à l'échelle, il est nécessaire de maintenir les historiques des souscriptions en mémoire centrale. Donc, nous cherchons une structure pour indexer ces historiques et un algorithme de traitement efficace des items.

Dans la première partie de ce chapitre, nous développerons la notion de système Pub/Sub, ainsi que différentes techniques d'indexation utilisées dans la littérature pour indexer les souscriptions, de différentes natures (mots-clés, expression booléennes). Et dans la seconde partie, nous présenterons des algorithmes et solutions proposés pour améliorer la qualité de l'information envoyée à l'utilisateur. Plusieurs solutions et approches ont été étudiées pour calculer le sous-ensemble le plus divers d'un ensemble donné : celles appliquées sur des ensembles statiques, ou bien sur des flux de données et en particulier pour les systèmes Pub/Sub.

2.2 Systèmes de Publication/Souscription

Les systèmes Publication/Souscription (ou Pub/Sub) sont des systèmes de notification où l'expéditeur (publieur) du message et le récepteur (souscripteur) ne sont pas synchronisés. L'expéditeur publie son message dans le système qui à son tour l'envoie aux souscripteurs concernés. Le souscripteur ne reçoit ainsi que les messages par lesquels il a exprimé son intérêt dans le système.

L'avantage des systèmes Pub/Sub est de rendre les publieurs indépendants des souscripteurs. Chacun d'eux peut continuer à fonctionner normalement même en l'absence de l'autre, contrairement aux systèmes traditionnels client/serveur : le client ne peut ni envoyer ni recevoir de messages du serveur lorsque celui-ci ne fonctionne pas. Les publieurs et les souscripteurs ne se connaissent pas directement.

Il existe deux familles de systèmes Pub/Sub : ceux basés sur des catégories ou « *Topic-based* » et ceux basés sur le contenu « *content-based* ».

(1) Système Pub/Sub basé sujet (*Topic-based*)
Dans le modèle *topic-based* [Ramasubramanian et al., 2006, Rowstron et al., 2001, Milo et al., 2007], les messages sont classés par groupe ou sujet. En publiant un message, le publieur doit lui attribuer une catégorie : le nom du sujet. Le souscripteur s'abonne aux sujets de messages auxquels il s'intéresse, des catégories comme le sport, les sciences ou la politique sur un site d'articles de presse.

(2) Système Pub/Sub basé contenu (*Content-based*)
Le modèle *content-based* est plus expressif grâce à l'attribution de champs supplémen-

taires [Aguilera et al., 1999, Carzaniga et al., 2001, Pereira et al., 2000a]. Les messages sont filtrés sur leur contenu en utilisant les critères de filtrage précisés par les souscripteurs. Les souscriptions sont des conjonctions de prédicats sur le contenu des messages [Aguilera et al., 1999]. Prenons par exemple un système de vente aux enchères sur Internet comme **ebay**. Un article à vendre est publié avec un prix, une catégorie, etc, et un souscripteur s'abonne à une catégorie d'articles avec un prix précis. Il sera notifié de l'existence de tout article répondant à cette souscription.

La richesse d'expression du système basé sur le contenu le rend d'autant plus complexe à gérer. Il faut préciser un schéma pour caractériser les messages et les souscriptions. En général, un schéma est composé d'un ensemble d'attributs avec leur domaine de valeurs dont il devient difficile de gérer les dimensions (*c.-à-d.*, les attributs).

À noter que les systèmes Pub/Sub basés sur le contenu sont plus flexibles que ceux basés sur les sujets et ne demandent pas de prédéfinition statique des sujets. De plus, les systèmes basés sur le sujet peuvent être facilement implantés à partir de ceux basés sur le contenu qui sont plus globaux [Aguilera et al., 1999]. Notre étude se focalisera exclusivement sur la seconde approche.

2.2.1 Indexation dans les systèmes Pub/Sub

Le problème de correspondance (*matching*) dans un système Pub/Sub consiste à trouver l'ensemble des souscriptions satisfaites par un événement produit [Aguilera et al., 1999] : une souscription est satisfaite par un événement si et seulement si les valeurs de ses prédicats vérifient la conjonction des prédicats de la souscription [Kale et al., 2005]. Deux types de prédicats qui sont utilisés pour les souscriptions sur le Web : textuel (mots-clés) ou structuré (paire attribut-valeur). Pour faciliter la recherche des souscriptions à notifier et afin de pouvoir maintenir un grand nombre de souscriptions, les systèmes Pub/Sub les indexent en mémoire. De plus, pour notifier les souscriptions en temps réel et supporter un débit élevé d'événements, le système doit appliquer un algorithme de recherche rapide et efficace dépendant de la structure d'indexation. Plusieurs techniques d'indexation des souscriptions en mémoire centrale sont présentées dans la littérature, nous allons les développer.

Listes inverses et Compteur

Dans cette approche les prédicats sont indexés dans des listes inverses, une liste est associée à chaque prédicat contenant l'ensemble des souscriptions dans lesquelles il apparaît. Pour trouver la correspondance d'un prédicat, il suffit de récupérer l'ensemble des souscriptions indexées dans une liste. Le problème se pose lorsque l'on souhaite récupérer des souscriptions composées de plusieurs prédicats. De fait, il est nécessaire de définir une structure qui vérifie pour chaque souscription, si la conjonction des prédicats a été vérifiée : le Compteur.

Ainsi, pour vérifier si un événement satisfait une souscription il est alors nécessaire de compter les prédicats de la souscription pour lesquels les valeurs sont validées. Un compteur est réservé pour chaque souscription. Pour chaque prédicat de l'événement, les compteurs des souscriptions concernées par ce prédicat sont incrémentés, puis les souscriptions dont la valeur du compteur égale à son nombre de prédicats à valider sont notifiées [Carzaniga and Wolf, 2003].

Le Subscribe [Fabret et al., 2000, Pereira et al., 2000a] indexe les prédicats dans un index basé sur le comptage, où les souscriptions sont traitées en comptant les prédicats contenus dans l'item. Le traitement des items est fait en deux étapes. Dans la première, les prédicats satisfaits sont cherchés en utilisant des index de prédicats qui les regroupent par attribut ou bien par type d'opérateurs. Toutefois, toutes les souscriptions satisfaites partiellement sont examinées, et quelques optimisations ont été proposées en regroupant les souscriptions selon leur taille suivant plusieurs variantes : en s'appuyant sur les prédicats de disjonction [Carzaniga and Wolf, 2003], en prenant en considération les statistiques sur les souscriptions et les items publiés [Fabret et al., 2001], ou bien en triant les identificateurs dans les listes des souscriptions dans le but de passer rapidement les souscriptions qui ne sont pas satisfaites [Whang et al., 2009].

Quelques travaux de la littérature se sont intéressés aux systèmes Pub/Sub pour des souscriptions de mots-clés. [Rose et al., 2007] présente COBRA un système de Pub/Sub basé sur le contenu. Ce système agrège les items des flux RSS et permet le filtrage de leur contenu. Les utilisateurs expriment leurs intérêts par des souscriptions sous forme d'un ensemble de mots-clés. Un système distribué est proposé pour le traitement des items publiés afin de trouver les souscriptions satisfaites par ces items. [Irmak et al., 2006] basé sur l'algorithme de comptage, prend en considération la distribution des occurrences des termes, dans le but d'optimiser le traitement des souscriptions par des techniques de regroupement . Le but de ce travail est de proposer un traitement efficace pour les index basés sur les listes inverses. Aucune comparaison de leur proposition avec une structure arborescente (comme le *Trie*) n'est présentée. Nous proposerons un système centralisé pour l'indexation des souscriptions pour la syndication Web (voir section 4.2). Nous présenterons également dans la section 4.5 une étude expérimentale approfondie pour étudier l'influence sur le comportement de ces index de plusieurs paramètres respectant les caractéristiques des données publiées sur le Web.

Dans les listes inverses, de nombreuses souscriptions sont traitées, et pourtant une grande majorité d'entre elles ne sont pas satisfaites. En effet, une souscription de n prédicats peut être satisfaite par $n-1$ de ses prédicats et ne pas être notifiée ce qui signifie $n-1$ tests inutiles. Plusieurs améliorations comme le regroupement des souscriptions par leurs prédicats [Pereira et al., 2000b] ou bien l'indexation par leur prédicat le plus sélectif (*c.-à-d.*, clé) [Yan and Garcia-Molina, 1994], ont été proposées dans la littérature pour les listes inverses dont le but est de réduire le nombre de tests effectués durant la recherche de souscriptions.

Afin d'éviter l'utilisation du compteur, [Yan and Garcia-Molina, 1994] propose un index

basé sur les listes inverses (*Ranked Key*) dont les profils utilisateurs sont représentés par des ensembles de mots-clés. Dans les listes inverses classiques, l'identificateur de la souscription est ajouté dans les listes de tous ses termes, alors que dans cette solution, il est ajouté seulement dans la liste du terme le moins fréquent des termes de la souscription suivi par la liste des termes restant. L'idée est que les termes très fréquents possèdent des listes courtes. L'avantage de cette idée est que le nombre moyen de souscriptions à traiter est moindre. En fait, les termes très fréquents dans les items auront des listes courtes qui seront parcourues plus pendant le traitement des items. Une étude complète de cet index sera présentée par la suite (section 4.2).

BRCQ (*Boolean Range Continuous Queries*) est un système récent proposé dans [Chen et al., 2013] pour évaluer les requêtes continues sur un flux d'objets textuels géolocalisé (*geo-textual objects*). L'index IQ-Tree (*Inverted File Quad-tree*) est proposé afin de supporter les grands débits de publications d'objets et d'arrivée de requêtes. IQ-Tree est un index hybride dans lequel un arbre est utilisé pour stocker l'information géographique des requêtes. Chaque zone géographique est caractérisée par ses codes de localisation et un rayon pour la dimensionner, et sera présentée par un nœud dans l'arbre. Pour chaque nœud une liste inverse est associée pour stocker le contenu textuel de la requête (l'ensemble de mots-clés) ainsi que son information temporelle (durée de la validité de la requête). Pour des raisons d'optimisation, ils utilisent la méthode (*Ranked Key*) proposée dans [Yan and Garcia-Molina, 1994]. Cet index est implanté sur le disque, les listes inverses sont stockées temporairement en mémoire. Quand la taille de la liste dépasse un seuil fixé, un sous-ensemble de requêtes y indexées est stocké sur le disque. Cette idée permet de minimiser le coût des entrées-sorties (*I/O*) vers le disque. Le sous-ensemble de requêtes est choisi selon l'information temporelle des requêtes et la fréquence de leurs mots-clés.

D'autres techniques s'intéressent à la factorisation de plages de valeurs des requêtes valuées. En effet, pour traiter un événement e, la solution naïve est de tester tous les prédicats indexés, pour le même attribut a. Mais plusieurs prédicats du même attribut peuvent toujours être satisfaits par les mêmes événements. Par exemple considérons les deux prédicats p_1 = (prix, $\leq$, 10) et p_2= (prix, $\leq$, 20). Nous pouvons remarquer que si p_1 est vérifié par un événement e alors il l'est également pour p_2. L'optimisation présentée dans [Pereira et al., 2000b] a pour but de minimiser le nombre d'évaluations de prédicats. Elle consiste à regrouper les prédicats indexés selon les opérateurs de comparaison et leur valeur. Chaque famille de prédicats sera indexée dans une structure de données. Dans la première étape de traitement d'un événement e, les ensembles de prédicats vérifiés par e sont cherchés, puis les compteurs des souscriptions associées aux prédicats vérifiés sont incrémentés. Enfin les souscriptions ayant une valeur de compteur égale au nombre de leurs prédicats sont notifiées. Chaque prédicat de l'événement n'est donc évalué qu'une seule fois pour un ensemble de souscriptions.

Dans [Shraer et al., 2013], un système de Pub/Sub basé sur une approche *top-k* est proposé pour annoter les historiques des nouvelles dans les médias sociaux. Les nouvelles sont considérées comme des souscriptions, et les *tweets* comme des items. Le score de

correspondance (*matching*) entre une nouvelle et un *tweet* est basé sur leur contenu, et un *matching* est reporté si ce score est supérieur au score du k-ième item déjà notifié pour une souscription donnée. Dans ce travail, les auteurs visent à trouver les historiques pour lesquelles chaque item publié se trouve parmi les *top-k* items classés. Toutefois, ce *matching* ne respecte ni le *broad* (inclusion de la souscription dans l'item) ni l'exact (égalité entre la souscription et l'item).

Par ailleurs, les listes inverses ont été exploitées dans [König et al., 2009] pour indexer les publicités (*advertisement bids*) proposées lors d'une recherche Web (*sponsored search*). Dans ce contexte, une requête soumise au système est évaluée avec l'ensemble des bids indexés. Un *bid* sera affiché si la requête contient tous ses termes. Afin de réduire l'espace de recherche de *bids*, les auteurs proposent un index multiterme basé sur les combinaisons des termes les plus fréquentes dans les *bids*. Le traitement d'une requête consiste alors à chercher dans l'index tous les sous-ensembles de mots-clés de la requête pour trouver l'ensemble de bids qui la satisfait. Comme le nombre de combinaisons croît exponentiellement avec la taille des requêtes, les auteurs fixent la taille maximale des combinaisons indexées. Cette optimisation ne permet pas cependant le passage à l'échelle pour les flux RSS où la taille des items publiés (en moyenne 25-36 termes) est largement plus grande que celle des requêtes sur le Web (en moyenne 2-3 termes [Beitzel et al., 2004]). Une autre optimisation est proposée dans le but de réduire le nombre d'accès en mémoire, elle consiste à réorganiser les publicités partageant le même sous-ensemble de termes. Les publicités sont indexées par le plus petit sous-ensemble (clé de hachage) et un nœud est associée à chaque publicité.

Arbres de correspondance

Les arbres sont des structures d'indexation souvent utilisés pour leur vitesse de recherche. Dans ces structures l'espace de recherche est fortement réduit. Faire une recherche à partir de la racine, en respectant les critères de recherche, permet d'éviter de parcourir tout sous-ensemble inutile de sous-arbres. Ainsi, cela limite la redondance d'information, car un seul chemin est utilisé pour indexer une même information (factorisation). Les arbres sont aussi utilisés pour indexer des phrases (ensembles de mots-clés) [Knuth, 1973]. Chaque nœud représente alors un mot-clé. Ils servent également à indexer des souscriptions sous forme de conjonctions de prédicats.

La technique présentée dans [Aguilera et al., 1999], transforme l'ensemble de souscriptions en un arbre qui sera utilisé dans la phase de recherche. Les souscriptions sont des conjonctions de prédicats, chaque prédicat représente un test booléen. Chaque nœud interne de l'arbre contient un test dont les résultats correspondent aux différents arcs sortant du nœud, et les feuilles sont les souscriptions. Ainsi, une souscription est représentée par la conjonction des prédicats de la racine vers une feuille.

Afin de résoudre le problème de correspondance (*matching*) dans les systèmes Pub/Sub basés sur les sujets, la structure arborescente présentée dans [Aguilera et al., 1999] peut

être adaptée pour l'indexation des souscriptions [Kale et al., 2005]. Chaque niveau de l'arbre représente un prédicat (sujet) et a au plus trois successeurs $\{0, 1, *\}$. Chaque feuille est une liste de souscriptions telle que le chemin de la racine vers cette feuille vérifie les valeurs de ses prédicats. Lorsque le souscripteur crée une souscription, s'il est intéressé par un sujet il met la valeur correspondante à ce sujet à 1 dans sa souscription, et à 0 sinon, et $*$ s'il est indifférent. Un événement est un ensemble de valeurs, une par sujet. La valeur est à 1 si l'événement contient des informations concernant le sujet, et à 0 sinon. Donc un événement est sous la forme $\{0, 1\}^k$ où k est le nombre de sujets. Un événement satisfait une souscription s'il a les mêmes valeurs pour les sujets par lesquels le souscripteur n'est pas indifférent (différents de $*$). L'événement doit contenir les sujets qui intéressent l'utilisateur, et ne pas contenir de l'information qu'il n'intéresse pas. Cependant, cette solution ne pourra pas être appliquée pour indexer les souscriptions pour la syndication Web à cause de la taille du vocabulaire publié sur le Web (plus de 1,5 million de termes section 3.5) qui affectera la profondeur de l'arbre. En utilisant le même modèle pour les souscriptions et les événements utilisé dans [Aguilera et al., 1999], [Kale et al., 2005] propose l'algorithme *RAPIDMatch*. Cet algorithme est basé sur le principe que beaucoup d'événements ont un petit nombre de prédicats pertinents pour lesquels ils doivent être traités. Les souscriptions sont indexées dans une structure arborescente basée sur des partitions à deux niveaux. D'abord, les souscriptions sont partitionnées selon le nombre de ses prédicats qui sont à 1 et ceux pour lesquels elle est indifférent $(*)$. Ensuite les souscriptions de chaque ensemble sont divisées par leurs prédicats dont les valeurs sont à 0 et $*$, en utilisant une partition déjà définie pour l'ensemble de prédicats. Cette stratégie de partitionnement permet de réduire l'espace de recherche de souscriptions et de déterminer rapidement les souscriptions satisfaites.

Dans [Yan and Garcia-Molina, 1994], les auteurs étudient, en plus des listes inverses, un arbre pour indexer des profils d'utilisateurs qui sont de petites tailles ($\sim$5 termes) et qui seront évalués sur de larges documents ($\sim$12 000 termes). Leur motivation est que des utilisateurs peuvent avoir des profils similaires, ayant des mots en commun. L'arbre est basé sur la factorisation des préfixes des profils indexés. Chaque nœud n correspond à un préfixe σ qui est l'ensemble de mots-clés du chemin de la racine jusqu'à n. Le nœud n contient la liste des profils ayant σ comme préfixe ainsi que leur suffixe et la liste de leurs fils. Cet arbre est implanté tel que la racine est le *dictionnaire* (l'ensemble de mots-clés des profils) sauvegardé en mémoire centrale et chacun de ses sous-arbres (ayant le mot m comme racine) qui correspond à l'ensemble des profils commençant par m, est sauvegardé sur le disque. Pour traiter un document, il faut lire en mémoire centrale les sous-arbres dont le mot de la racine appartient au document. Chacun sera parcouru en appliquant l'algorithme de parcours en largeur d'un arbre. Pour chaque nœud visité, les fils dont le mot apparaît dans le document sont ajoutés à la liste de nœuds à traiter. Ensuite, la liste des profils de chaque nœud est parcourue afin de tester l'inclusion de suffixe des profils dans le document. Si le suffixe est vide ou s'il est inclus dans le document, le profil est considéré comme satisfait.

Pour prendre en considération tous les types d'opérateurs (d'égalité (=) et d'inégalité

($\leq$ *et* $\geq$)) dans des espaces multidimensionnels, des index arborescents ont été utilisés dans [Wang et al., 2004, Machanavajjhala et al., 2008]. Dans ce type d'index l'espace de recherche (les souscriptions) est réduit à des points, et les événements publiés à des intervalles de requêtes. Dans [Wang et al., 2004], le UB-Tree est utilisé pour passer d'un espace multi-dimensionnels à un espace unidimensionnel. Puis les intervalles sont indexés dans un arbre B. L'évaluation expérimentale de leur index montre qu'il est de quatre ordres de magnitude plus rapide que l'algorithme de comptage. [Machanavajjhala et al., 2008] étend cet index pour les prédicats conjonctifs dans les systèmes Pub/Sub basés sur le classement des résultats. Le but de cet index est de chercher les souscriptions les plus pertinentes pour un item donné et non uniquement celles satisfaites par le *broad match*. Pour cela un index unidimensionnel de type *Scored Interval Index* est utilisé par dimension dans le but de supporter un *matching* partiel entre les souscriptions et les items.

Une structure de données arborescente dynamique pour indexer les expressions booléennes (*Boolean Expressions*) est présentée dans [Sadoghi and Jacobsen, 2011]. Les auteurs proposent le BE-Tree (*Boolean Expression-Tree*) qui permet le passage à l'échelle avec des millions d'expressions booléennes dans un espace multi-dimensionnel. BE-Tree est un index basé sur l'attribut clé (*key-based index*) implanté en mémoire centrale ce qui lui permet une meilleure performance que les index proposés dans [Aguilera et al., 1999, Fabret et al., 2001, Whang et al., 2009, Yan and Garcia-Molina, 1994]. Cet index se base sur une approche à deux phases. La première phase consiste à partitionner l'espace des attributs afin de déterminer le meilleur attribut pour le partitionnement. La seconde phase est un regroupement dans la structure locale de chaque partition des expressions par des intervalles des valeurs de l'attribut de la partition. Lorsque le nombre d'expressions de chaque groupe dépasse sa capacité, un autre partionnement par attribut des expressions y indexées est fait. Une version améliorée nommée BE*-Tree est proposée dans [Sadoghi and Jacobsen, 2013], avec une étude analytique du temps nécessaire pour ajouter une souscription, le temps de traitement et l'espace mémoire consommé. Cependant, le temps nécessaire pour ajouter une souscription ne dépend pas seulement de la taille de la souscription (nombre de prédicats par souscription), contrairement à notre index basé sur les arbres *Regular Ordered Trie* (voir section 4.2), mais aussi de la cardinalité du domaine (taille du vocabulaire dans notre contexte). Un système Pub/Sub *top-k* basé sur BE$*$-Tree est proposé dans [Sadoghi and Jacobsen, 2012] pour une recherche efficace des souscriptions satisfaites par un item publié. Les sous-arbres sont parcourus par l'ordre de pertinence de la clé correspondante à leur racine à l'item publié, puis l'algorithme de recherche par profondeur est appliqué pour parcourir les sous-arbres. Le processus de traitement s'arrête lorsque les *top-k* expressions sont trouvées, ou lorsque toutes les partitions pertinentes de l'arbre sont parcourues.

Autres index pour les Pub/Sub

Dans la littérature, il existe des systèmes Pub/Sub qui ont été proposés en se basant sur d'autres structures que les listes inverses et les arbres comme par exemple des structures

basées sur les graphes et les *clusters* (groupes). Nous présentons les plus caractéristiques ci-après.

Approches basées sur les graphes Une technique d'indexation est présentée dans [Shen et al., 2005] en se basant sur la relation de couverture des souscriptions afin d'augmenter l'efficacité des systèmes de notification qui ont des architectures distribuées. Soient deux souscriptions s_1 et s_2 : s_1 couvre s_2 si et seulement si toutes les notifications qui satisfont s_2 satisfont aussi s_1. Dans ces systèmes, chaque routeur gère un ensemble de clients et est lié à deux autres routeurs. Un client soumet sa souscription au routeur auquel il est attaché. Le routeur la transmet aux routeurs voisins, et elle sera propagée de proche en proche, de sorte que tous les routeurs ont une connaissance de toutes les souscriptions soumises au système. En recevant un événement, le routeur la transmet aux routeurs qui lui ont transmis la souscription correspondante. Le problème de cette approche est que le routeur doit avoir connaissance de toutes les souscriptions. Ils est donc nécessaire d'avoir une grande table de routage ce qui ralentit les notifications. Afin de réduire la taille de cette table, ils profitent de la relation de couverture entre les souscriptions. L'idée d'utiliser la relation de couverture consiste à ne pas transmettre toutes les souscriptions aux autres routeurs : si une nouvelle souscription est couverte par une autre souscription déjà existante elle ne sera pas transmise. Mais quand une souscription s est supprimée, il faut couvrir les souscriptions s_i qui sont couvertes par s à l'aide d'autres souscriptions. S'il n'y en a pas il faut transmettre les s_i qui ne sont pas couvertes. La relation de couverture est représentée par un graphe, dans lequel un arc direct entre s_i et s_j existe si et seulement si s_i couvre s_j. Ainsi une souscription sera transmise si elle ne possède pas d'arcs entrants (*c.-à-d.*, elle n'est pas couverte par une autre souscription). Le graphe peut avoir de nombreux arcs, la difficulté est alors de le maintenir. Pour cela, ils proposent une alternative qui consiste à limiter le nombre d'arcs entrants par souscription, ce qui implique qu'elle ne sera éventuellement pas liée à toutes les souscriptions qui la couvrent.

[Petrovic et al., 2005, Liu et al., 2008] proposent un système de Pub/Sub basé sur les graphes pour filtrer les flux RSS. Ils représentent les souscriptions comme des n-tuplets dans un graphe orienté et étiqueté. Chaque souscription est sous la forme (sujet, propriété, objet, contraintes (sujet), contraintes (objet)). Les sujets et les objets sont présentés par des nœuds liés par des arcs étiquetés par la propriété correspondante. Toutes les souscriptions sont fusionnées dans un seul graphe indexé en utilisant des tables de hachage à deux niveaux. Tous les sommets du graphe sont indexés dans la première table de hachage (les noms des sommets sont pris comme clé de hachage) et chaque entrée est un lien vers une autre table de hachage qui contient la liste d'arcs entre ces nœuds. Chaque entrée de la deuxième table de hachage est un lien vers la liste de souscriptions. Une analyse théorique de l'espace mémoire utilisé par l'index ainsi que le temps de traitement est présentée. L'espace mémoire consommé dépend du nombre d'arcs dans le graphe ainsi que le nombre moyen de souscriptions associées à chaque arc. Le temps de traitement est linéaire avec le nombre de souscriptions satisfaites, ce qui n'est pas le cas pour nos

index où le temps de traitement est indépendant de nombre de souscriptions satisfaites (voir section 4.3).

En se basant sur un automate non-déterministe fini, [Demers et al., 2006] propose un système de Pub/Sub pour le traitement des souscriptions avec des flux d'événements comme les flux RSS. Les publications sont considérées comme des événements temporels avec un temps de départ et temps de fin. Les souscriptions sont expressives en contenant des opérateurs relationnels (*par ex.*, *when*, *where*, *only*, *first* ...). Malgré son expressivité, ce système ne permet pas le passage à l'échelle avec millions de souscriptions. Il a été évalué pour 400 000 de souscriptions indexées et d'après les résultats le temps augmente exponentiellement avec le nombre de souscriptions.

Approches basées sur les clusters Les systèmes Pub/Sub basés sur les sujets (*topic-based*) sont souvent utilisés dans les applications de communications. Ils sont considérés comme des canaux de communication dans lesquels chaque utilisateur précise les sujets qui l'intéressent. Avec le grand nombre de sujets sur le Web et le débit élevé des événements, la diffusion des événements devient très coûteuse en temps de transmission des événements et de maintenance des index. Dans ces systèmes, les sujets sont séparés et chacun est géré indépendamment, bien que nombreux sujets aient un nombre important de souscriptions en commun.

Afin de minimiser le coût de maintenance des systèmes distribués pour les Pub/Sub basés sur les sujets, [Milo et al., 2007] propose de regrouper les sujets similaires ayant presque les mêmes souscripteurs mais traités séparément. La solution est de fusionner ces sujets en *topic-cluster*, et les souscripteurs déclarent leurs intérêts au système qui doit déterminer l'ensemble de sujets (*topics*) ou bien groupe de sujets (*topic-cluster*) auxquels il le souscrit. Il y a donc moins de redondance des souscriptions, d'où une réduction de la taille de la structure de données et un coût de maintenance réduit. Les auteurs proposent un modèle probabiliste pour estimer le bénefice à ajouter un sujet à un groupe ou bien de souscrire un utilisateur à un sujet ou bien groupe. Il dépend du nombre de souscripteurs associés au sujet et au groupe ainsi que la fréquence de publication pour un sujet donné. Leur proposition peut être appliquée pour les index distribués de type *multicast tree* dans le but de réduire le coût de sa maintenance.

Cette approche est aussi utilisée pour indexer les souscriptions dans les systèmes basés sur le contenu, [Fabret et al., 2000] propose de regrouper les souscriptions sur leur prédicat d'égalité : deux souscriptions avec des prédicats d'égalité pour les mêmes attributs sont placées dans le même groupe. Une souscription est placée dans un seul de ces groupes. Les prédicats d'inégalité sont stockés de la même manière que dans un algorithme basé sur le compteur. Après avoir testé la satisfaction des prédicats d'égalité de chaque groupe, les prédicats d'inégalité de chaque souscription sont vérifiés grâce à l'algorithme basé sur le compteur.

Ces méthodes de regroupement ne peuvent pas être appliquées pour des souscriptions composées de conjonction de mots-clés. Les termes n'ont pas de plages de valeurs, ils

sont présents ou non dans les souscriptions. Les index basés sur la clé de la souscription (*key-based*) se basent sur la même idée pour minimiser le nombre de tests à effectuer (plus de détails dans la section 4.2).

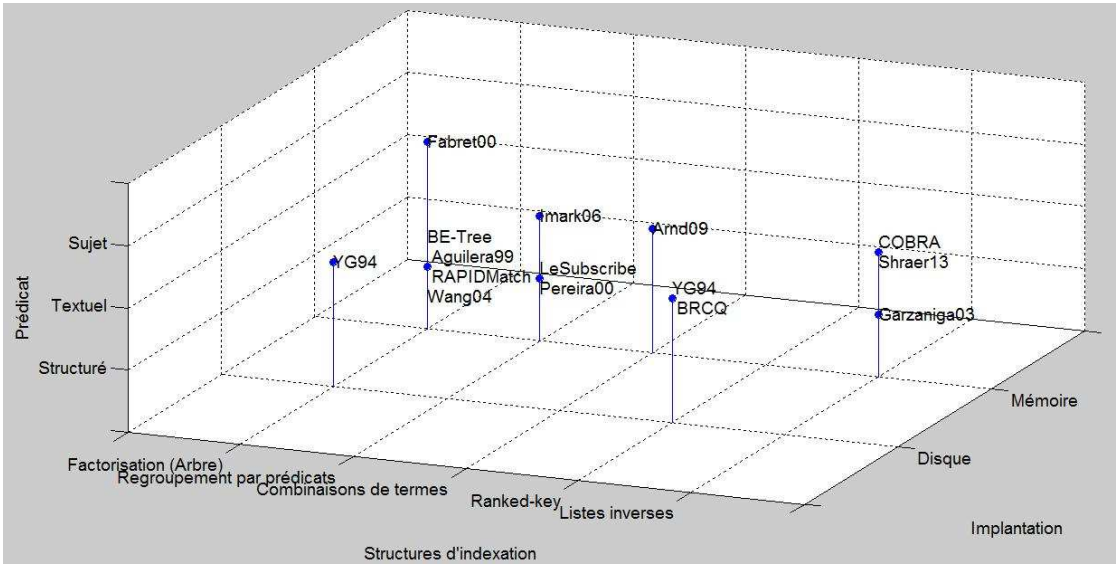

Figure 2.1 – Indexation dans les systèmes Pub/Sub

Dans cette thèse, nous nous intéressons aux index de types listes inverses et arbres basés sur le contenu textuel (mots-clés). Dans le diagramme de la figure 2.1, nous présentons les principaux travaux de recherche présentés précédemment. Nous les avons classés selon trois axes : structure d'indexation (listes inverse vs. *Trie*), implantation (disque vs. mémoire) et prédicat structuré vs. regroupement par sujet. Nous constatons que la majorité des travaux sont implantés en mémoire en centrale afin d'offrir des recherches rapides. De plus, un grand nombre de solutions sont dédiées aux données structurées (paires attribut-valeur). Nous pouvons également observer que la plupart des structures s'intéressent au regroupement et à la factorisation des prédicats. Dans le chapitre 4, nous allons proposer des index en mémoire centrale pour des souscriptions qui sont des ensembles de mots-clés.

2.2.2 Comparaison des structures d'indexation

Certains travaux de recherche ont comparé ces structures d'indexation, dans des contextes différents du nôtre (Web, *sponsored search*), en étudiant par exemple leur taille en mémoire et en évaluant leur performance en temps de recherche, comme par exemple [Yan and Garcia-Molina, 1994]. Dans le chapitre 4, nous présenterons une étude expérimentale approfondie pour comparer ces structures (les listes inverses et les arbres). Nous étudierons l'impact de différents paramètres du Web (taille de souscriptions, distribution de fréquences des termes...) sur leur morphologie. Nous comparerons leurs

performances en espace mémoire et temps de traitement pour le passage à l'échelle avec de larges vocabulaires et des millions de souscriptions.

Dans un système basé sur les listes inverses avec compteur, pour un événement e, et pour chaque propriété p_i de e, le compteur de toutes les souscriptions satisfaites par p_i est mis à jour. Dans le cas des arbres, après l'évaluation de p_i seules les souscriptions qui ont les i propriétés satisfaites seront considérées Ainsi les souscriptions non satisfaites sont supprimées rapidement de l'espace de recherche et le nombre de souscriptions traitées est inférieur à celui des systèmes basés sur le compteur. Observons que l'algorithme basé sur les arbres incrémente implicitement une fois le compteur pour un ensemble de souscriptions, ce qui réduit le nombre de mises à jour du compteur.

[Fabret et al., 2000] compare l'algorithme de *Gough* [Gough and Smith, 1995] basé sur les arbres avec celui basé sur le compteur en optimisant le nombre d'évaluations de prédicats. Ce dernier est plus efficace en mémoire car il n'y a pas de redondance de prédicats comme dans un arbre où le prédicat est stocké plusieurs fois et la souscription peut correspondre à plusieurs chemins. En effet, en fonction de sa fréquence, un prédicat peut se retrouver dupliqué plusieurs fois dans un arbre. D'après notre étude de la morphologie des index (section 4.5.3), nous avons pu mettre en valeur le gain de la factorisation entre les souscriptions pour minimiser la redondance des termes dans l'arbre où les souscriptions correspondent à un seul chemin.

Les expériences faites dans [Pereira et al., 2000b] qui comparent l'algorithme basé sur le compteur avec celui des *clusters* avec un prédicat caractéristique, montrent que les deux approches sont capables de manipuler un grand nombre de souscriptions, et que l'utilisation du prédicat le plus souvent choisi rend l'algorithme des *clusters* plus performant. L'utilisation d'un prédicat sélectif qui caractérise la souscription permet de réduire le nombre de souscriptions à traiter. Ainsi, le *Ranked key Inverted List* est basé sur ce principe pour améliorer les performances des listes inverses classiques basées sur le compteur. Notre comparaison de ces index montre le gain obtenu en temps de traitement pour les index basés sur la clé de la souscription.

Un petit nombre de systèmes Pub/Sub ont été proposés pour les souscriptions de mots-clés. SIFT [Yan and Garcia-Molina, 1999] est utilisé pour la diffusion de documents textuels. Une variante a été étudiée en détail dans [Yan and Garcia-Molina, 1994]. Ces études sont basées sur des implantations disques du : (a) *Ranked Key Inverted List* dans lequel les souscriptions sont indexées dans la liste de son terme ayant le rang le plus petit, (b) *Regular Trie* et (c) *Regular Ordered Trie* (ROT), mais elles ne se sont pas intéressées au POT (notre variante de *Patricia Trie* (voir chapitre 4)). Il est attendu que l'utilisation d'un ordre total des termes permette d'avoir plus de préfixes en commun entre les souscriptions. Les auteurs ont trouvé que le traitement du ROT prend significativement plus de temps que les index basés sur les listes inverses car les documents ont besoin de plus d'entrées/sorties (I/O) pour être lus. Toutefois, nous avons proposé une implantation en mémoire centrale ce qui améliore les performances en passant à l'échelle avec un large nombre de souscriptions (environ 10 millions). Les principales différences

entre leurs observations et les nôtres sont dues à (a) la taille en nombre de termes des événements publiés (52 en moyenne pour la syndication Web [Hmedeh et al., 2011b] vs. 12 000 pour les documents), (b) la taille du vocabulaire des souscriptions (1,5 millions pour la syndication Web vs. 18 000 pour les profils), et (c) la distribution des occurrences des termes.

2.3 Nouveauté et diversité de l'information

Dans cette thèse, nous nous intéressons également au filtrage de l'information par nouveauté et diversité. Le but de ce filtrage est de satisfaire au mieux l'utilisateur en ne lui envoyant que de nouvelles informations, sans redondance et tout en couvrant des domaines variés. Dans cette section, nous allons présenter un état de l'art sur les travaux de la littérature qui proposent des solutions afin de favoriser la nouveauté et la diversité de l'information. Nous présentons d'abord la définition du problème de la nouveauté et de la diversité, ainsi que les algorithmes proposés pour une recherche efficace et pertinente du résultat. Par la suite, nous classons les solutions proposées selon leur contexte.

2.3.1 Définitions et algorithmes

La `nouveauté` de l'information a été étudiée dans le contexte de la recherche d'information et des systèmes de recommandations dans le but d'éviter d'envoyer des informations redondantes à l'utilisateur. Une information est redondante si elle est couverte par une autre déjà reçue par l'utilisateur [Zhang et al., 2002].

La `diversité` a été étudiée dans la littérature pour satisfaire les utilisateurs en leur envoyant un ensemble d'items pertinents par rapport à leur requête, tout en étant différents. La diversité de l'information peut être considérée comme l'inverse de la similarité : moins les résultats d'un ensemble sont similaires plus son information est diverse (ce qui revient à augmenter la distance moyenne entre les éléments de cet ensemble). Elle a été utilisée pour répondre aux requêtes dans de larges bases de données [Vee et al., 2008], classer les résultats renvoyés par les moteurs de recherche [Clarke et al., 2008, Carbonell and Goldstein, 1998], ou par les systèmes de recommandations [Abbar et al., 2013, Yu et al., 2009a, Yu et al., 2009b] et filtrer des items dans les systèmes Pub/Sub [Drosou et al., 2009, Pripužić et al., 2008].

Plusieurs définitions de la diversité des items sont présentées dans la littérature en se basant sur : (1) le contenu, (2) la nouveauté , ou (3) la couverture [Drosou and Pitoura, 2010]. Dans notre travail, nous nous intéressons à la diversité en se basant sur le contenu des items afin de notifier des éléments pertinents, non redondants (nouveaux) et globalement peu similaires (divers). Cette distinction a été faite dans le contexte de la recherche d'information [Clarke et al., 2008], où la nouveauté a été utilisée pour supprimer la redondance et la diversité pour résoudre le problème de l'ambiguïté.

Cependant, la majorité des travaux définissent le problème de la diversité comme un problème *top-k*; étant donné un ensemble d'éléments S (solution) qui satisfait une requête, le problème est de trouver les k éléments qui favorisent le plus la diversité. Par rapport à notre contexte de flux RSS, un petit nombre de travaux considère que l'ensemble d'items change au cours du temps (flux d'items), ils calculent la diversité sur un ensemble statique d'items.

La solution naïve pour résoudre le *top-k* consiste à calculer toutes les combinaisons possibles de k éléments puis choisir l'ensemble ayant la plus grande distance moyenne entre ses éléments. Dans le but de ne pas utiliser cette solution très coûteuse, il existe deux familles d'algorithmes qui sont souvent utilisées pour sélectionner les k éléments les plus distants : *Greedy* et *Swap*.

L'ensemble des éléments résultats dans l'algorithme *Greedy* [Agrawal et al., 2009, Gollapudi and Sharma, 2009] est initialisé par l'élément le plus pertinent à la requête et à chaque itération l'élément le plus distant, en moyenne, aux éléments précédemment choisis est ajouté au résultat. L'algorithme s'arrête lorsque k éléments ont été ajoutés. Une autre version de cet algorithme initialise l'ensemble de résultat par les deux éléments les plus distants à la requête et à chaque itération les deux éléments les plus distants sont choisis [Gollapudi and Sharma, 2009]. Le but de cette proposition est de réduire le nombre d'itérations à faire pour choisir l'ensemble d'éléments le plus distant.

Dans l'approche *Swap* [Yu et al., 2009a, Liu et al., 2009], l'ensemble résultat est initialisé par k éléments, ces éléments peuvent être les k éléments les plus pertinents à la requête ou bien choisis au hasard. À chaque itération, un de deux éléments les plus similaires dans l'ensemble du résultat est remplacé par l'élément le plus distant à cet ensemble. Une autre solution possible pour choisir le nouvel élément à échanger est de prendre l'élément qui augmente le plus la distance moyenne entre les éléments de l'ensemble de résultat. L'algorithme s'arrête s'il n'y a plus d'éléments dans l'ensemble solution qui dépassent un seuil de pertinence à la requête.

Dans le cadre de la diversité dans les flux de données, une fenêtre glissante (en temps ou bien en nombre d'éléments) est utilisée pour limiter le nombre d'éléments à prendre en considération pour calculer l'ensemble résultat qui favorise la diversité. Il est également possible de faire un traitement périodique sur l'ensemble d'éléments publiés entre deux instants. Toutefois, il est nécessaire de souligner que dans le contexte des systèmes Pub/Sub, un élément déjà notifié ne peut pas être enlevé du résultat. Pour cette raison, nous ne pouvons pas utiliser les algorithmes d'échange (*Swap*) avec nos hypothèses, et les approches incrémentales (*Greedy*) doivent être appliquées en prenant en considération l'ensemble d'éléments déjà notifiés à l'utilisateur.

2.3.2 Domaines et approches

Dans cette section, nous présentons plusieurs travaux de recherche pour calculer la nouveauté et la diversité pour des ensembles statiques ou des flux de données. Nous présen-

tons également des travaux pour le filtrage par diversité dans les systèmes Pub/Sub. Nous allons voir que la nouveauté et la diversité ont été également combinées avec d'autres critères de filtrage, comme la pertinence des éléments à la requête.

Ensemble statique d'éléments

L'objectif des moteurs de recherche est de présenter à l'utilisateur une liste de documents pertinents à sa requête choisis parmi les documents indexés dans leur base. Un défi des moteurs de recherche est de satisfaire au mieux l'utilisateur, et ils doivent ainsi lui présenter au moins un document qui satisfait ses intérêts. [Carbonell and Goldstein, 1998] propose de prendre en considération la diversité dans les moteur de recherche de l'information tetuelle. Il introduit la méthode *Maximal Marginal Relevance*. Cette approche combine la notion de pertinence et la diversité des documents pour calculer des scores et ainsi classer les documents. Plus un document a un rang élevé dans le résultat (*c.-à-d.*, classé en premier), plus il est similaire à la requête tout en étant moins similaire aux documents de rang plus élevé. L'ensemble des documents ainsi créé est construit d'une manière incrémentale.

Avec le grand volume de données et l'ambiguïté des requêtes soumises par les utilisateurs, il est devenu difficile aux moteurs de recherche de raffiner les intérêts des utilisateurs. De petites requêtes ne sont pas suffisantes pour exprimer spécifiquement l'information que l'utilisateur recherche. De plus, l'utilisateur a tout intérêt à recevoir des informations diverses. Il n'est pas satisfait s'il reçoit plusieurs informations concernant un sujet provenant de la même source ou bien traitant une partie du domaine qu'il cherche. Pour résoudre le problème de l'ambiguïté des requêtes soumises sur le Web, [Clarke et al., 2008] propose un modèle probabiliste pour classer les documents en prenant en considération leur nouveauté et diversité.

Pour augmenter la diversité de l'information envoyée à l'utilisateur en éliminant l'information redondante, [Angel and Koudas, 2011] mesure l'importance d'un document comme étant le produit de sa pertinence par rapport à la requête et sa nouveauté (la probabilité que son contenu ne contienne pas de l'information redondante). Pour éviter de calculer la pertinence du document à la requête, puis sa nouveauté, ils proposent l'algorithme *DivGen* pour un traitement en une seule phase. L'objectif de ce travail est d'envoyer à l'utilisateur les k documents les plus pertinents à la requête et les moins similaires entre eux.

Les traitements en une seule phase ne peuvent pas être appliqués au système basé sur le filtrage par une inclusion totale de la requête dans le document (*broad match*). Les traitements en une seule phase cherchent un compromis entre la pertinence et la diversité par un seul calcul. Et la sémantique du *broad match* n'attribue aucun seuil de pertinence entre la requête et le document. Il est nécessaire de vérifier la satisfaction de la requête par le document avant de tester la diversité de ce dernier. Nous verrons dans le chapitre 4 comment nous cherchons d'abord les souscriptions satisfaites par les items publiés, puis

nous proposons un filtrage par nouveauté et diversité indépendant de *matching* entre les souscriptions et les items dans le chapitre 5.

Dans [Yu et al., 2009a, Yu et al., 2009b], les auteurs prennent en considération la diversité de l'information pour répondre aux requêtes dans les systèmes de recommandation. L'idée consiste à présenter à l'utilisateur les k items les plus distants. La distance entre deux items est basée sur leur *explanation*. L'*explanation* d'un item est un sous-ensemble des items déjà notifiés à l'utilisateur qui lui sont similaires et bien classés par l'utilisateur. Cette approche est utile quand il n'y a pas d'attributs en commun entre les items ou quand il n'existe pas d'attribut descriptif pour comparer les items.

Afin de favoriser la diversité dans les systèmes de recommandation en se basant sur les commentaires des utilisateurs, dans [Abbar et al., 2013] les auteurs étudient plusieurs approches pour calculer la diversité (K_{NN}, MMR [Carbonell and Goldstein, 1998]). Ils proposent un algorithme incrémental (*Greedy*) très efficace pour calculer les *top-k* articles et qui favorise la diversité. Plusieurs techniques pour mesurer la distance entre les articles sont élaborées : (1) celles basées sur les entités présentes dans les commentaires (sujets et noms des entités), (2) celles basées sur les sentiments en utilisant un dictionnaire pour calculer le score de l'article se basant sur les mots négatifs et positifs de ses commentaires, (3) celles qui considèrent l'ensemble des utilisateurs qui ont participé aux discussions de l'article, et (4) celles qui prennent en considération la localisation des utilisateurs qui ont participé aux discussions. Leur étude qualitative montre que le calcul de la diversité en se basant sur les sentiments dans les commentaires des utilisateurs est un bon compromis entre la pertinence et la diversité des articles choisis.

[Smyth and McClave, 2001] étudie l'impact de la valeur de k sur la diversité et la similarité de *top-k* items choisis dans les systèmes de recommandation. Le fait d'augmenter la diversité entre les items réduit leur similarité à la requête. Ils proposent un compromis entre la diversité et la similarité. L'impact du choix de k surtout avec des taux de notifications différents pour les souscriptions, rend une approche *top-k* peu appropriée pour les flux RSS.

Nous proposons un filtrage pour les systèmes Pub/Sub sans limiter le nombre d'items à envoyer à l'utilisateur à k. Notre but est qu'un utilisateur reçoive toute nouvelle information publiée sans être limité à k éléments, ou bien qu'il soit notifié par des éléments plus que le nécessaire pour couvrir la nouvelle information afin d'atteindre les k éléments.

Une autre approche basée sur la diversité ne limitant pas la taille de l'ensemble d'items résultat à k est présentée dans [Drosou and Pitoura, 2012a]. Les auteurs proposent un index basé sur un graphe d'items pour sélectionner le nombre minimal d'items représentatif du contenu de l'ensemble solution. Cet ensemble doit contenir pour chaque item de la solution un item qui lui est similaire, toutefois les items choisis ne doivent pas être très similaires. Chaque item est représenté par un nœud dans le graphe, et un arc existe entre deux nœuds si et seulement si la distance entre les items qui leur correspondent est inférieure ou égale à un rayon fixé. Regardant la première condition pour choisir l'ensemble résultat qui consiste à avoir pour chaque item un item qui lui est similaire, elle est

satisfaite en choisissant un ensemble dominant du graphe. Soit D l'ensemble dominant d'un graphe G, D est un sous-ensemble des nœuds de G tel que tout nœud de G qui n'est pas dans D est lié à au moins un nœud de D. La seconde condition est accomplie en choisissant un ensemble indépendant I de G qui est un sous-ensemble des nœuds de G tel qu'il n'existe pas deux nœuds de I qui soient liés c'est-à-dire non similaires. En utilisant ces définitions, l'ensemble d'items à présenter est équivalent à l'ensemble indépendant dominant minimal du graphe.

Flux de données

La nouveauté a été également utilisée comme critère de sélection dans les systèmes adaptatifs de filtrage d'information [Zhang et al., 2002]. Contrairement aux travaux déjà présentés [Carbonell and Goldstein, 1998, Angel and Koudas, 2011], la solution proposée ne se base pas sur une combinaison de pertinence et de nouveauté. Les auteurs proposent un traitement en deux étapes : premièrement la pertinence du document à la requête est calculée, puis il est filtré par sa nouveauté. Notre proposition pour le filtrage des items des flux RSS est similaire à cette approche, d'abord les items sont filtrés par leur pertinence aux souscriptions (*broad match*) (chapitre 4) puis par leur nouveauté et leur diversité (chapitre 5). Cependant, pour définir l'ensemble de documents à prendre en considération dans leur filtrage, ils se basent sur un historique de dix derniers items envoyés à l'utilisateur. Cette approche peut générer des comportements de filtrage différents lors de fortes pics de variation des flux RSS (section 3.3.2) Les documents sont présentés par des ensembles de mots avec redondances, et plusieurs métriques pour mesurer la distance entre eux sont étudiées. Ils proposent *Set difference* pour mesurer la distance entre les documents. Elle est basée sur la fréquence des termes dans les items et le nombre d'items déjà notifiés qui les contiennent. Ils trouvent que Cosinus (mesure symétrique), qui est appropriée pour mesurer la distance entre les documents avec redondance de termes (*bag of terms*), est meilleure pour mesurer la redondance. Ce résultat n'est pas attendu car la redondance est théoriquement une mesure asymétrique ce qui n'est pas le cas pour Cosinus. Par contre, Cosinus ne peut pas être utilisée pour mesurer la distance entre nos items qui sont de petits ensembles de termes distincts. Nous proposons dans la section 5.4 une mesure asymétrique basée sur l'importance globale des termes dans les items pour mesurer leur nouveauté. En plus, nous trouvons que sur le Web l'ordre de publication des items est très important pour calculer leur nouveauté d'où le besoin d'une mesure asymétrique.

Dans [Drosou and Pitoura, 2012b], les auteurs proposent un index dynamique pour la diversité des données continues. Ils utilisent une fenêtre glissante sur le temps pour limiter l'ensemble d'items à prendre en considération pour calculer la diversité du nouvel item publié. L'idée est de présenter à l'utilisateur les k items les plus distants dans la fenêtre. L'approche proposée prend en considération la durabilité et la fraîcheur des items de la fenêtre. Un nouvel item publié qui ne favorise pas la diversité n'est supprimé de la fenêtre que s'il y a un autre item plus récent qui est parmi les k items les plus distants

dans la fenêtre. Cette hypothèse respecte l'ordre chronologique des items pour prendre en considération la fraîcheur de l'information. La durabilité des items garantie que les items jugés comme divers et déjà envoyés à l'utilisateur restent parmi les items choisis dans les fenêtres sortantes. Ces items sont utilisés pour initialiser l'ensemble des items divers. Dans un autre travail [Drosou and Pitoura, 2009], l'ensemble d'items restant dans la fenêtre est utilisé pour initialiser l'ensemble dans l'algorithme *Greedy*. Leur index est basé sur un arbre couvrant (*cover tree*) [Beygelzimer et al., 2006] qui permet l'insertion et la suppression des items dans un environnement dynamique en se basant sur des algorithmes incrémentaux. Chaque nœud correspond à un seul item qui peut être présenté par plusieurs nœuds dans l'arbre (un nœud par niveau). C'est un arbre ascendant dont le dernier niveau contient tous les items. Pour chaque nœud au niveau n correspond un nœud père au niveau $n-1$ qui le couvre. Ainsi, l'ensemble des k items les plus divers peut être sélectionné parmi les items des niveaux élevés. De plus, cet index permet de répondre à plusieurs requêtes avec différentes valeurs de k.

Dans le cadre du filtrage par nouveauté et diversité (chapitre 5), nous nous basons également sur une fenêtre glissante sur le temps, mais nous utilisons une autre approche pour gérer notre fenêtre glissante dans la section 5.4. Nous ne gardons pas dans la fenêtre les items non notifiés (c.-à-d., pertinents par le *broad match*, mais filtrés par nouveauté et/ou diversité) à l'utilisateur même s'ils font parti de la période de la fenêtre.

Une approche incrémentale pour la diversité des requêtes continues sur un flux d'items est proposée dans [Minack et al., 2011]. Le but de cette approche est de choisir les k items les plus divers à un instant donné. Ils proposent de prendre en considération le temps de publication des items pour calculer leur pertinence à la requête. Cette hypothèse est justifiée par le fait que l'utilisation d'une fenêtre de taille fixe n'est pas appropriée pour les différents taux de publication des items. En effet dans la même fenêtre, les items les plus anciens qui sont très pertinents à la requête vont impacter le score des nouveaux items qui leur sont similaires.

Dans [Panigrahi et al., 2012], les auteurs présentent une méthode pour le filtrage par diversité dans un flux d'items. Chaque item est associé à plusieurs caractéristiques qui disposent d'un objectif équivalent à la couverture désirée. Le but est de maximiser la plus petite couverture des caractéristiques dans les items choisis. Ils proposent un filtrage en deux étapes : d'abord un filtrage global est fait pour tous les utilisateurs, puis un autre filtrage est réalisé en prenant en considération les intérêts de chaque utilisateur. L'objectif du premier filtrage est de choisir un ensemble représentatif de B items des items publiés. Ce filtrage est global et indépendant des requêtes soumises. Ils utilisent un algorithme de seuil qui prend en considération l'objectif accompli pour les caractéristiques dans les items sélectionnés afin de réduire leur importance en sélectionnant d'autres items. Il est plus efficace que d'autres algorithmes basés sur des seuils fixes ou sur des choix aléatoires qui se basent sur les fréquences des caractéristiques. Malgré le résultat approximatif calculé, leurs expériences montrent qu'il est meilleur que les résultats fournis par les autres algorithmes même pour le cas de la caractéristique la moins couverte.

Un système de Pub/Sub qui prend en considération la diversité de l'information envoyée à l'utilisateur est proposé dans [Drosou et al., 2009]. Une solution incrémentale est proposée comme pour les solutions statiques (*Greedy*) pour choisir les k items les plus distants. L'ensemble résultant est initialisé par les deux items les plus distants, et incrémentalement l'item le plus distant aux items déjà choisis est ajouté au résultat. Dans ce système, les auteurs prennent en considération les préférences de l'utilisateur. Pour cela, la distance entre les items prend en compte les scores donnés par l'utilisateur pour les différents attributs de sa souscription. Plusieurs types de distribution sont étudiés pour définir l'ensemble d'items sur lequel la diversité des items est calculée. Trois modes sont présentés : (1) périodique (*top-k* sur des périodes disjointes de temps de taille fixe), (2) fenêtre glissante (calculer les *top-k* quand un nouvel item est publié, qui est envoyé à l'utilisateur s'il fait parti de *top-k* items) et (3) filtrage basé sur l'historique (quand un nouvel item est publié, ils décident de l'envoyer à l'utilisateur en se basant sur les *top-k* derniers items envoyés à l'utilisateur sur une fenêtre de w items). Dans le mode basé sur une fenêtre glissante, un item qui n'est pas envoyé à l'utilisateur dans une fenêtre w_i peut être parmi les *top-k* items dans une prochaine fenêtre w_{i+1}. Dans le mode basé sur l'historique, le filtrage est basé sur l'algorithme *Swap*, c'est-à-dire l'item est envoyé à l'utilisateur si le fait de l'échanger avec un item de *top-k* items reçus par l'utilisateur augmente la diversité de l'ensemble d'items envoyés à l'utilisateur.

Afin de résumer ces travaux, nous les avons classés par domaine et type d'algorithme appliqué, et calcul de diversité dans la figure 2.2. Nous trouvons que l'algorithme incrémental est souvent utilisé grâce à son efficacité de calcul par rapport à celui basé sur les échanges. Nous remarquons que la majorité des travaux ont combiné le calcul de la diversité avec le calcul de la pertinence. Par contre, dans notre système de filtrage pour les souscriptions des flux RSS, nous séparons la phase de *matching* (pertinence souscription-item) de celle du filtrage par nouveauté et diversité de l'information publiée dans les items.

2.4 Conclusion

Dans ce chapitre nous avons présenté plusieurs approches et des algorithmes pour la notification dans les systèmes Pub/Sub. Toutefois, aucune ne permet de répondre efficacement à des souscriptions composées de mots-clés à l'échelle du Web sur la syndication Web. Pour cela, nous adapterons trois structures classiques pour indexer les souscriptions pour les flux RSS et nous étudierons l'impact des paramètres du Web sur leurs performances. Nous avons également présenté des travaux de recherche qui proposent des solutions pour favoriser la nouveauté et la diversité de l'information envoyée à l'utilisateur. Les solutions proposées pour le Pub/Sub sont souvent des solutions basées sur une approche *top-k*. Nous proposerons par la suite un filtrage par nouveauté et diversité basé sur une fenêtre glissante sur le temps pour les items publiés par les flux RSS.

Plusieurs travaux de recherche ont étudié les caractéristiques des données publiées

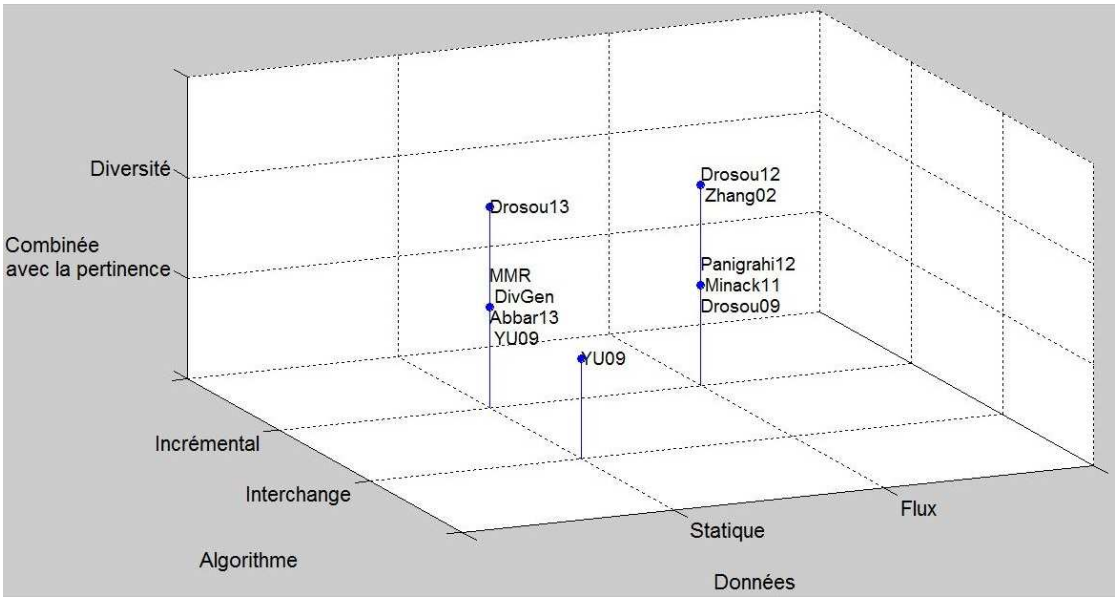

Figure 2.2 – Techniques de filtrage par diversité

sur le Web. Il existe cependant peu de travaux dédiés à une étude précise du comportement de flux RSS et de leur contenu [Liu et al., 2005, Thelwall et al., 2006, Lambiotte et al., 2007, Oita and Senellart, 2010]. Dans [Liu et al., 2005], les auteurs ont étudié l'activité des flux. Ils ont trouvé que 57% des flux publient de nouvelles informations toutes les deux heures. [Thelwall et al., 2006] propose une classification de l'activité des flux selon les pics d'activité. [Oita and Senellart, 2010] basé sur une analyse de 400 flux, conclue que seul un petit nombre d'items ont de grandes tailles. Finalement, concernant la caractérisation du vocabulaire des données publiées sur le Web, [Williams and Zobel, 2005] ont étudié 5,5 millions de documents du Web pour lesquels ils ont extrait 10 millions de termes distincts. Ils ont trouvé qu'une loi de *Heap* caractérise l'évolution de la taille du vocabulaire, et que ses paramètres dépendent du jeu de données utilisé ainsi que du processus d'extraction des termes. Le même comportement a été observé pour les requêtes sur le Web [Zien et al., 2001]. [Williams and Zobel, 2005, Sia et al., 2007, Hu and Chou, 2009] ont aussi étudié la distribution des fréquences des termes et l'ont caractérisée par une loi *Zipf*. D'autres travaux ont caractérisé les fréquences des termes avec d'autres ditributions. [Montemurro, 2001] a proposé une distribution *Zipf-Mandelbrot*, et [Lambiotte et al., 2007] a suggéré une distribution *Power Law* modifiée. [Silverstein et al., 1999, Jansen and Pooch, 2001, Spink et al., 2001] montrent que plus de 50% des termes des requêtes sont utilisés une seule fois, et qu'environ 80% apparaissent moins que trois fois. Pour étudier la co-occurrence des termes, [König et al., 2009] présente des statistiques sur les *sponsored searches*, la co-occurrence des termes suit une distribution *Zipf*.

Ces études réalisées pour caractériser les flux RSS ne sont cependant pas suffisantes pour

concevoir et paramétrer des applications Web (agrégateurs, systèmes de notifications) passant à l'échelle. Ainsi l'activité des flux RSS (taux de publication) n'a pas été étudiée ni leur variation en fonction du temps, deux paramètres qui ont un impact très important sur le débit d'entrée du système. Il est également important de connaître la distribution des tailles en nombre de termes des items à évaluer afin de prévoir la capacité du système pour des traitements à la volée sur le Web. Une étude du contenu sémantique des items est aussi intéressante pour améliorer la qualité de l'information diffusée aux utilisateurs. Cependant des études détaillées sur le vocabulaire publié (taille, évolution) par les flux RSS, ou sur l'évolution des rangs de ses termes n'existent pas. Une telle étude est importante pour les index basés sur les rangs des termes notamment pour les concevoir dynamiques et flexibles aux variations sur le Web. La composition du vocabulaire impacte également le débit de notifications dans les systèmes de filtrage alors que la distribution des occurrences des termes influe-elle sur les notifications, d'où le besoin d'adaptation des approches de mesure de satisfaction.

Avant d'étudier des techniques d'indexation pour les systèmes Pub/Sub, nous allons donc nous intéresser au comportement des flux RSS sur le Web afin de caractériser les données manipulées et orienter nos choix de structures et d'implantations.

Chapitre 3

Caractérisation des flux RSS

3.1 Introduction

Les sites Web diffusent de plus en plus de nouvelles informations publiées sur leurs pages en utilisant les formats de la syndication Web tels que les flux RSS [RSS, 2003] ou Atom [Atom, 2007]. Malheureusement, les études faites pour caractériser les flux RSS/Atom [Liu et al., 2005, Thelwall et al., 2006, Lambiotte et al., 2007] ne montrent pas des statistiques précises sur leur activité et le contenu de flux pouvant permettre d'améliorer la performance des agrégateurs de flux et des systèmes de notifications des utilisateurs. À cette fin, nous avons souhaité pousser cette analyse de comportement selon trois axes principaux : taux de publication des flux, caractéristiques de leurs items, et leur contenu textuel.

Notre étude a été effectuée sur un jeu de données réel qui a été collecté durant une période de huit mois à partir de mars 2010. Cela nous a permis de collecter 10 794 285 items publiés par 8 155 flux productifs (provenant de 2 930 sites différents). Le jeu de données et ses sources ainsi que son stockage dans une base de données seront détaillés dans la première partie de ce chapitre. Ensuite, nous présenterons une étude caractérisant les flux RSS basée sur leur type de source et leur taux de publication. Pour les items publiés par ces flux, nous étudierons leur structure XML et leur taille en nombre de termes. Pour finir dans la dernière partie, nous étudierons le contenu textuel de ces items avec la taille de leur vocabulaire et son évolution avec le nombre d'items publiés, ainsi que l'évolution des rangs des termes.

3.2 Description du jeu de données

Les sites Web utilisent les flux RSS [RSS, 2003] pour diffuser les nouvelles informations publiées sur leurs pages. Les fréquentes mises à jour des pages Web peuvent être considé-

rées comme des listes suivies par l'utilisateur. Les standards flux RSS sont basés sur des fichiers de format XML qui permettent aux sites Web de diffuser les nouvelles informations publiées comme étant des listes de liens vers le contenu publié. Les sites Web ont besoin de garder un flux (*feed*, ou bien *channel*), sur leur page pour permettre aux utilisateurs de suivre les actualités. Un flux contient une liste d'`items`, chacun est identifé par un lien. Les items sont des résumés de l'information publiée sur le Web et peuvent aussi inclure quelques *méta-données* en plus. La figure 3.1 présente un exemple d'un flux RSS avec ses méta-données (*par ex.,* dans la balise *channel* (flux) : *title*, *link*, *language*, *pubDate*) et les informations de ses items (*par ex.,* dans la balise *item* : *title*, *link*, *description*, *pubDate*, *guid*, *category*).

```
<?xml version='1.0' encoding='UTF-8' ?>
<rss version="2.0">
    <channel>
        <title>Mon exemple de flux</title>
        <link>http ://www.exemple.org</link>
        <language>fr</language>
        <lastBuildDate>Mer, 03 Jan 2012 13 :36 :10 GMT</lastBuildDate>
        <item>
            <title>Mon premier item</title>
            <link>http ://www.exemple.org/rss.html ?item=1</link>
            <description>
                Ce text décrit l'item, et donne plus d'information du contenu de lien.
            </description>
            <pubDate>2012-01-03 13 :24 :10</pubDate>
            <guid isPermaLink="false">http ://www.exemple.org/rss/item1.xml</guid>
            <category>Exemple</category>
        </item>
        ...
    </channel>
</rss>
```

Figure 3.1 – Exemple d'un flux RSS

Une fois qu'un flux est disponible sur une page Web, les applications Web (agrégateurs) peuvent régulièrement accéder au fichier XML du flux pour obtenir les items les plus récents. Pour simplifier, un flux RSS est un lien qui renvoie un document XML selon un format RSS qui contient une liste d'items résumant son contenu principal. Atom [Atom, 2007] est un autre format standard de la syndication Web normalisé par l'IETF [1]. Bien que les deux formats Atom et RSS aient quelques différences de structures, ils possèdent les mêmes fonctionnalités, et beaucoup de gens utilisent, par abus de langage, le terme « RSS » pour désigner RSS ou bien Atom sans les différencier.

Pour notre étude, nous avons extrait 100 000 flux de différentes sources de flux : annuaires

1. Internet Engineering Task Force

RSS/Atom, portails et moteurs de recherche (comme : syndic8.com, completeRSS.com, Google Reader, feedmil.com, retronimo.com ou bien thoora.com) sur une période de huit mois à partir de mars 2010. D'après la catégorisation de notre jeu de flux (section 3.3.1), nous pouvons remarquer qu'il peut être considéré comme représentatif de l'ensemble des flux RSS du Web.

Parmi ces flux, nous avons pu constaté que seulement 12 611 ne contenaient pas d'erreurs de communication et validaient le format RSS/Atom. Les autres flux ont été supprimés pour une des raisons suivantes : (a) lien vers le flux n'existe plus ou bien a été modifié, (b) accessibilité possible uniquement à partir d'une session utilisateur, (c) erreurs ou bien des balises manquantes du format XML. De plus, pendant la période de huit mois, quelques flux n'ont produit aucun items, par conséquent notre étude est ramenée à 8 155 flux. Ces flux ont publié 27 003 115 items qui ont été stockés dans une base de données MySQL [2] parmi lesquels nous avons extrait les flux sans duplication d'items donnant 10,7 millions d'items.

Toutes les expériences de notre étude ont été effectuées sur notre base de données, par extraction de l'information correspondante (*par ex.,* taux de publication de flux, catégories, taille d'items). Le schéma de la base de données détaillé dans le tableau 3.1 a été conçu pour stocker les différents attributs des flux et des items pour faciliter l'extraction de toutes les informations nécessaires pour nos analyses. Nous avons représenté toutes les balises standards des flux RSS/Atom dans notre schéma (voir tables `Feed` et `Item`). Les tables `FeedCategory` et `ItemCategory` ont été définies dans le but de catégoriser les flux et les items. Les identificateurs des flux et des items sont associés à un identificateur d'une catégorie de la table `Category`. Quelques tables sont ajoutées pour stocker les informations non standards (`FeedExtension` et `ItemExtension`), comme nous le verrons dans la section 3.4, 29,73% des items possèdent des balises non standards. Nous remarquons aussi une utilisation incorrecte des balises standards que nous avons stocké dans ces tables (*par ex.*, balises Atom dans des flux RSS). Pour finir, une table spécifique, `Pack`, a été définie pour stocker l'information temporelle de l'acquisition des items ; à chaque fois qu'un nouvel ensemble d'items est acquis. Cette information est utilisée pour calculer l'activité des flux RSS en taux de publication d'items. Par conséquence, les items de la table `Item` sont identifiés par le triplet (`FeedId, PackNum, ItemNum`) ; l'attribut `GUID` n'a pas été retenu pour identifier les items car la balise qui lui correspond n'existe que dans 69% des items (voir section 3.4).

La date de l'item, les termes et la catégorie sont des paramètres importants pour nos statistiques et la caractérisation des flux, nous attirons l'attention sur quelques points importants :

– La date de publication (`pubDate`) est absente dans 20% des items, nous avons donné un *timestamp* pour chaque item qui correspond à sa date d'acquisition. Il est calculé comme étant le temps passé depuis l'insertion du premier item dans notre base (voir table `Pack` dans le schéma). Cela nous garantira pour des utilisations ultérieures (sec-

2. Disponible en ligne sur deptinfo.cnam.fr/∼traversn/roses

Nom de la Table	Attributs	Description
Feed	FeedId, URL, link, Title, GUID, Description, Language, Author, TTL, skipHours, skipDays, LastPubDate, Contributor, Generator, Logo, Icon, Copyright, Rating, Cloud, Docs, WebMaster, textInput, Version	Toutes les balises standards des formats RSS et Atom
FeedCategory	FeedId, Category	Association entre un flux et quelques catégories
Category	CategoryId, Name	Un identificateur unique par nom de catégorie
FeedExtension	FeedId, Num, tag, attributes, namespace, Extension, path	Balises non standards ajoutées aux flux
Pack	FeedId, PackNum, Timestamp, NbItem, nbNewItem	Paquet d'items acquis temporellement
Item	FeedId, PackNum, ItemNum, GUID, PubDate, Link, Title, Description, Content, Lang, Author, Source, Enclosure	Toutes les balises standards d'un item des formats RSS et Atom, lié à un paquet d'item
ItemCategory	FeedId, PackNum, ItemNum, Category	Association entre un item et quelques catégories
ItemExtension	FeedId, PackNum, ItemNum, NumExt, tag, attributes, namespace, Extension	Balises non standards ajoutées aux items

Table 3.1 – Schéma de base de données du jeu de données

tions 3.3.2 et 3.5.4) une reproduction réaliste du flux d'items ;

– Dans le but de caractériser la répartition des flux en terme de source d'information selon leur type (`Social Media, Press, Forums, Sales, Misc. et Blogs`), nous nous appuyons sur une classification manuelle des flux en se basant sur : l'information de leurs méta-données (balises : titre, catégorie, lien ou bien description) par la recherche des mots-clés spécifiques (*par ex.*, *currencies*, *forum*, *daily news*) et leur hyponymes dans *WordNet*[3], domaines de leur site d'hébergement (*par ex.*, *twitter*, *blogpost*, *yahoo*), ou bien compositions particulières des items (*c.-à-d.,* un modèle unique pour tous les items) ; [Oita and Senellart, 2010] considèrent huit domaines pour classifier leurs flux dédiés au domaine de la science et rapportent le temps moyen de mises à jour pour chaque domaine. Notre étude s'applique sur un jeu de données plus large ($\sim$ 12 611 flux) ce qui nous permet de classer nos flux sur des domaines plus généraux ;
– Pour chaque item, nous ne gardons qu'une seule occurrence. Afin de détecter les items

3. wordnet.princeton.edu

dupliqués, nous avons étudié la correspondance exacte entre les items en utilisant une fonction de hachage sur leur contenu textuel (titre et description). Grâce à ce filtrage, nous avons éliminé quelques flux très redondants qui ne respectaient pas l'hypothèse de l'ordre chronologique par re-publication de leurs items (domaines d'actualité), ou bien qui changent leur contenu automatiquement selon un modèle (nombre d'utilisateurs qui ont vu les items, rang courant des items) ;

– Nous avons aussi filtré automatiquement les flux (items) écrits en anglais, l'absence de l'information concernant la langue des flux a rendu cette tâche difficile. 7 691 008 items anglais ont été manuellement extraits, comme suit : un item est considéré comme un item anglais si sa balise de langue (*language tag*) existe dans les méta-données du flux elle est « en », ou bien le lien du flux est dans un site d'hébergement anglais (*par ex.*, « us » ou bien « uk »), ou bien l'item appartient à un flux de lien « .com » sans aucune balise de langue. Puis, une analyse lexicale du contenu textuel des items extrait des balises : `titre` et `description`, a été effectuée en utilisant des outils standards de stemming (comme `SnowBall`[4] - en se basant sur l'algorithme de Porter [van Rijsbergen et al., 1980]) et dictionnaires (comme `WordNet`) pour la langue anglaise. Nous avons extrait les termes des items et réalisé une étude statistique détaillée dans la section 3.5.

3.3 Analyses des flux

Dans cette section, nous allons étudier les caractéristiques de notre jeu de données regardant le type en source d'informations des flux agrégés, ainsi que leur activité en taux de publication. Des études statistiques concernant l'activité des applications Web 2.0 ont déjà été effectuées[5], mais des études plus fines concernant l'activité des flux RSS/Atom en prenant en considération leur type n'ont pas été réalisées dans la littérature. Comme le taux de publication des flux les plus actifs est prévisible, nous pourrons lier leur activité à leur type de source.

3.3.1 Types des flux

Nous avons identifié six différents types pour les sources des flux : `Press` (pour les journaux et les agences), `Blogs` (pour les blogs personnels), `Forums` (pour les forums de discussion et les listes de diffusions), `Sales` (pour les sites Web de commerce), `Social media` (pour les micro-blogging comme twitter, digg, yahoo ! groups, et blogosphere) et `Misc.` (pour les sites web d'information avec des renseignements médicaux/sur les villes, ainsi que des podcasts). Les 8 155 flux actifs ont été classés manuellement comme nous avons pu le voir dans la section 3.2.

4. snowball.tartarus.org
5. thefuturebuzz.com/2009/01/12/social-media-web-20-internet-numbers-stats

Type	**% des flux**	**% des items**	$\frac{\#\ items}{\#\ flux}$
Social Media	1,77%	9,45%	7085,03
Press	9,99%	38,82%	5141,24
Forums	1,51%	3,62%	3178,01
Sales	11,32%	15,49%	1811,92
Misc.	41,47%	25,47%	812,99
Blogs	33,93%	7,14%	278,55

Table 3.2 – Types de sources des flux RSS/Atom

Le tableau 3.2 présente pour chaque type : son pourcentage en flux et items ainsi que le nombre moyen d'items par flux. `Social media`, `Press` et `Forums` sont très productifs (avec un nombre moyen d'items par flux entre 3 178,01 et 7 085,03) puis `Sales`, `Misc` et `Blogs` (avec moins de 2 000 items en moyenne par flux). Comme nous pourrons le voir dans les sections suivantes, le comportement des flux peut être affiné en tenant compte à la fois du taux de publication quotidien, et de la variance de leur activité en fonction du temps.

Nous trouvons que la composition de notre jeu de données est représentatif du contenu et de la réalité du Web 2.0. Par exemple, le fait que les flux de type `Blogs`, par opposition à ceux de `Social media` qui ont un taux de publication bas par rapport à leur nombre, la disponibilité effective des flux de `Press` et `Sales` a un bon impact sur la composition de notre jeu de données. Plus précisement, le tableau 3.3 présente un ensemble des *URL*s par classe d'activité, ces classes et leur caractéristiques seront détaillées par suite.

3.3.2 Activité des flux

Nous nous sommes intéressés à l'activité des flux, aussi bien en terme de débit, mais également variation de celui-ci au cours du temps afin de mieux appréhender la productivité des flux qui représente le débit d'entrée de notre système de Publication/Souscription.

Concernant l'activité de flux, nous avons observé une distribution *Power Law* des flux par rapport à leur taux de publication ce qui confirme les résultats de [Sia et al., 2007]. Cette loi s'explique par un petit nombre de flux (8%) sont très productifs (publient 84% des items) et que la majorité des flux (71%) sont peu productifs. Le taux de publication moyen est 3,59 items par jour (Voir figure 3.2). Mais elle ne permet pas de prendre en considération la variation du débit de publication pour la caractérisation de l'activité des flux.

Ainsi, nous avons étudié l'activité des flux en mesurant le nombre de pics d'activité (*burstiness*) produits par le flux pendant une période de temps (un jour). Pour cela, nous avons utilisé la définition de *burstiness* de [Thelwall et al., 2006], un flux produit un *burst* si durant une courte prériode de temps (jour) il publie un nombre d'items 5 fois plus grand que son taux de publication moyen par jour. Toutefois, alors que 89% de nos

Classe d'activité	URL	type	taux	coeff.
`Productive`	rss.sportsblogs.org/rss/daily.xml	Social	1275	0,008
	feeds.feedburner.com/Techcrunch	Social	25	0,012
	press.jrc.it/rss ?id=all.rss&type=bns	Press	1184	0,009
	preciseNews.us/pt/planetrss ?group=news	Press	471	0,009
	feeds.feedburner.com/fixya/wgvY	Misc	593	0,008
`Moderate`	publisherdatabase.com/forums/rss.php ?t=1	Forum	2,26	0,029
	csmonitor.com/rss/usa.rss	Press	9,6	0,017
	lemonde.fr/rss/sequence/0,2-3234,1-0,0.xml	Press	9,4	0,027
	blog.reidreport.com/feed	Blog	5,31	0,016
	medicalnewstoday.com/rss/cancer-oncology.xml	Misc	8,88	0,021
`Slow`	tugsearch.co.uk/blog/feed	Blog	0,90	0,048
	koantum.wordpress.com/feed	Blog	0,10	0,993
	download-soft.com/rss-feeds/new-Audio.xml	Sales	0,96	0,597
	fashionjewelryforeveryone.com/Under5ERRSS.xml	Sales	0,86	0,993
	feeds.ezinearticles.com/expert/Smit_Chacha.xml	Misc	0,88	0,114

Table 3.3 – Exemples des URLs par classe d'activité

flux produisent des *bursts*, les 11% restant produisent plus que 81% des items. Nous en concluons donc que cette mesure ne permet pas de caractériser le comportement de nos flux, en particulier les très productifs.

A cette fin, nous avons alors utilisé le cœfficient de *Gini* (G) [Pitoura and Triantafillou, 2008] pour caractériser la variation des activités de publication de flux au cours du temps. Il est adéquat pour les séries temporelles (les flux sont des séries temporelles), il ne prend pas seulement en considération la déviation de l'activité des flux mais aussi la variation au cours de temps de cette déviation. Il a été principalement utilisé dans plusieurs domaines comme l'économie, la géographie, l'ingénierie, ou l'informatique. Le cœfficient de *Gini* est défini comme suit :

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |y_i - y_j|}{2 \times n \times \sum_{i=1}^{n} y_i} \tag{3.1}$$

avec y_i est égal au nombre d'items par jour, classés par leur ordre décroissant sur n jours. Une valeur du cœfficient de *Gini* proche de 1 signifie que le flux subit une variation significative au cours du temps.

La figure 3.3 montre en échelle log-log, le cœfficient de *Gini* en fonction du taux de publication moyen de chaque flux. Les flux ayant une valeur de $G < 0,02293$ (au-dessous de la ligne horizontale pointillée) ne produisent aucun pics pendant toute la période de 8 mois. De ces résultats, nous avons extrait trois classes d'activité : la première `Productive` composée de 614 flux qui produisent plus que 10 items par jour avec une variation temporelle très basse ($G < 0{,}03$) ; la deuxième `Moderate` de 1 677 flux qui publient entre 1 et

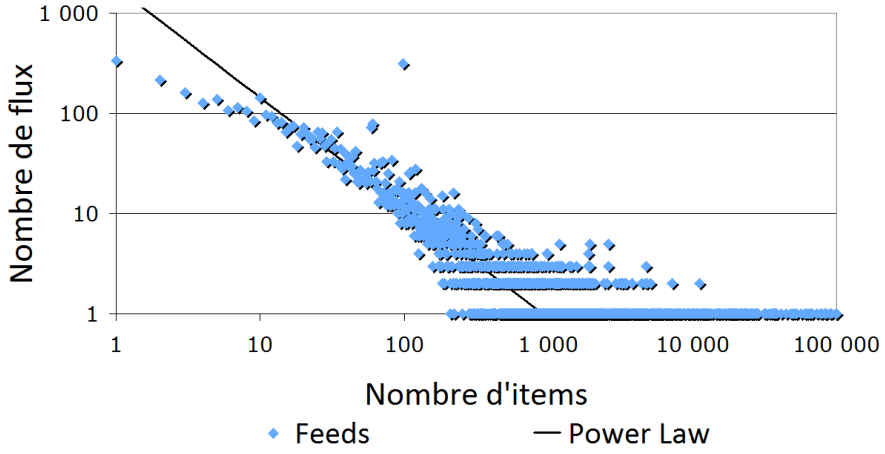

Figure 3.2 – Nombre de flux par nombre d'items publiés

10 items par jour avec une variation temporelle raisonnable ($G < 0{,}1$) ; la dernière `Slow` représente la majorité des flux qui publient moins d'un item par jour et produisent une forte variation temporelle en nombre d'items ($G = 0{,}32$ en moyenne).

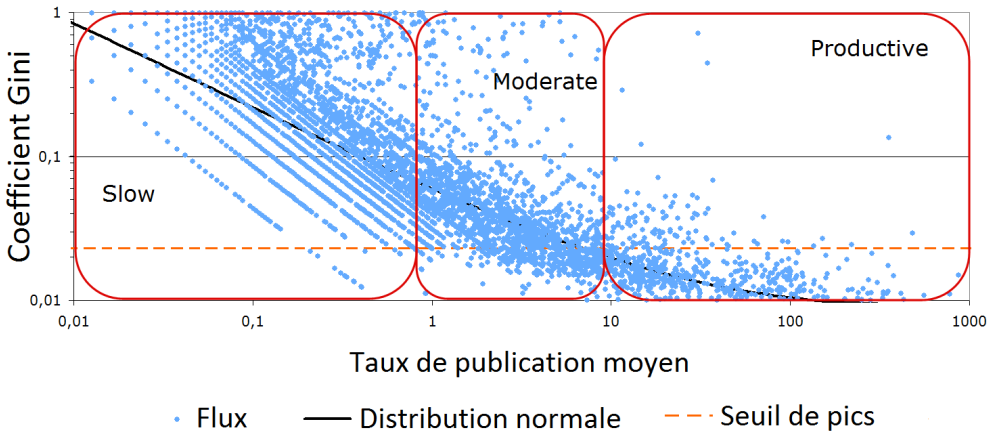

Figure 3.3 – Classes d'activité de flux RSS/Atom

Dans le but d'étudier la variation des flux en prenant en considération leur taux de publication p, nous avons formellement caractérisé la distribution de la valeur moyenne de G (courbe continue dans la figure 3.3). Cette distribution est caractérisée par une ligne droite qui termine par une longue queue dans une échelle log-log. L'équation 3.2 a été utilisée pour caractériser cette courbe avec une distribution *Power Law* modifiée. Nous avons obtenu une valeur du test d'adéquation (*Chi-square*) de 229,5 avec 200 en degré de liberté, pour les valeurs suivantes pour les constantes : $\alpha_1 = 5,53 \times 10^{-2}, \theta_1 = -0,59, \alpha_2 = 4,48 \times 10^{-3}, \theta_2 = 0,101$. Cette fonction est la seule qui a été capable de

montrer que nous avons une très petite variance de publication pour les flux ayant un taux de publication élevé (longue queue).

$$G(p) = \alpha_1 \times p^{\theta_1} + \alpha_2 \times p^{\theta_2} \tag{3.2}$$

Le tableau 3.4 montre pour chacune de ces classes son taux de publication moyen, variance, covariance, nombre maximal d'items et valeur moyenne de *Gini*. À noter que les flux qui produisent de *bursts* (flux de type *Blogs* et *Forums*) ont une valeur de G dix fois plus élevée que les flux productifs. La variance de taux de publication est très élevée pour les flux de la classe `Productive`. En fait, cette classe a plusieurs pics avec différentes amplitudes, mais la valeur de cœfficient de *Gini* correspondante est petite, qui donne à une petite variation en taux de publication au cours du temps. Contrairement aux flux de la classe `Slow` qui ont une petite variance et valeur moyenne élevée de cœfficient de *Gini*, illustrant leur comportement en pics occasionnels.

	`Productive`	`Moderate`	`Slow`
Taux de publication moyen	61,816	3,592	0,308
Variance	14241,24	5,72	0,062
Covariance	4,71x 10^{-8}	3,96 x 10^{-9}	5,47 x 10^{-10}
Nombre maximal d'items	11 314	12 235	64
Cœfficient *Gini* moyen	0,0204	0,0596	0,1321

TABLE 3.4 – Caractérisation de différentes classes d'activité

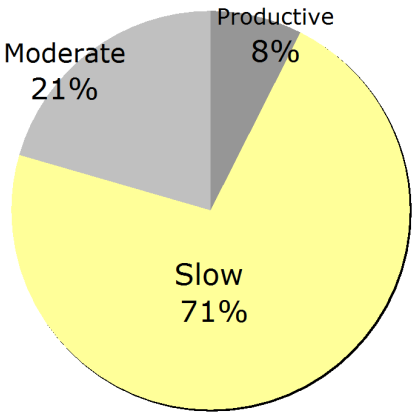

FIGURE 3.4 – Pourcentage de flux par classe d'activité

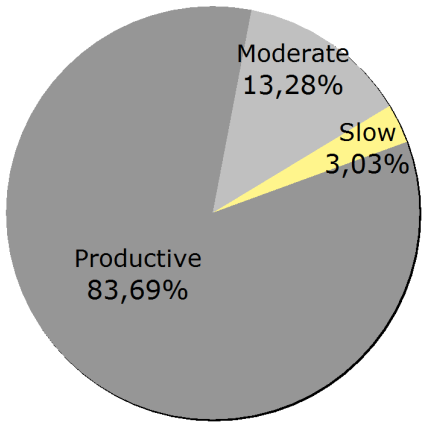

FIGURE 3.5 – Pourcentage d'items par classe d'activité

Contrairement à [Liu et al., 2005] qui concluent que 57% des flux ont un taux de publication supérieur à 24 items par jour, dans la figure 3.4 nous remarquons que seulement 8% de nos 8 155 flux qui sont très productifs (plus de 10 items par jour), ont publié la majorité de nos items (83,69% des items) tandis que 71% ont un taux de publication

bas. À noter aussi que parmi les 12 611 flux agrégés environ 35% n'ont publié aucun items pendant la période de huit mois. Nous pouvons aussi remarquer à partir de la figure 3.5, que la majorité des flux de la classe `Slow` produisent seulement 3,03% des items (distribution *Power Law* pour le taux de publication des flux).

Pour finir, le tableau 3.5 montre pour chaque combinaison de type de flux et classe d'activité : le pourcentage du nombre de flux, le taux de publication moyen (p), ainsi que, la valeur moyenne du cœfficient de *Gini* (G). Les flux de type *Social Media* sont répartis entre les deux classes `Productive` et `Slow` avec une variation temporelle de leur taux de publication plus importante que les autres dans la classe `Slow`. Ceci est lié au comportement des publieurs, par exemple dans *Twitter* le comportement des *Followers* et des *Followees* (les utilisateurs peuvent suivre - classe `Slow` - ou bien être suivis et produisent beaucoup - classe `Productive`). Les flux de type *Press* suivent un comportement modéré, alors que les flux de type *Forums*, *Sales*, *Misc* et *Blogs* font partis de la classe `Slow`. Il faut également remarquer que dans cette classe, *Sales* a une variance de publication importante tandis que *Blogs* et *Forums* ont un comportement régulier.

Type	`Productive`			`Moderate`			`Slow`		
	%	p	G	%	p	G	%	p	G
Social Media	**44,9%**	65,33	0,015	8,1%	5,57	0,026	**47,0%**	0,18	0,470
Press	27,4%	70,76	0,020	**43,0%**	3,61	0,036	29,6%	0,34	0,237
Forums	21,0%	54,51	0,018	20,2%	3,85	0,042	**58,8%**	0,29	0,348
Sales	8,3%	83,64	0,022	19,0%	3,43	0,122	**72,7%**	0,23	0,454
Misc	2,7%	63,12	0,024	12,9%	3,70	0,059	**84,4%**	0,23	0,338
Blogs	4,3%	15,53	0,019	24,7%	3,37	0,056	**71%**	0,22	0,267

Table 3.5 – Types de source et classes d'activité des flux

Dans cette section, nous avons pu distinguer trois classes d'activité des flux RSS : `Productive`, `Moderate` et `Slow`. Cette classification prend en compte le taux de publication des flux ainsi que la variation de leur publication. Nous avons trouvé que les flux productifs ont une variation très basse en taux de publication au cours du temps. Contrairement, aux flux de la classe `Slow` qui publient moins d'un item par jour, produisent une grande variation. Ce comportement d'activité est bien lié au type de flux. Nous avons vu que les flux de type *Social Media* sont répartis entre les deux classes `Productive` et `Slow`, dû à l'activité des publieurs et récepteurs. Les flux de type *Press* publient leurs items avec un comportement régulier (majorité font partis de la classe `Moderate`). Les autres types de flux (*Sales*, *Misc.* et *Blogs*) ne produisent pas beaucoup d'items, mais avec une variation plus importante pour le type de *Sales*.

3.4 Analyses des items

Après avoir caractérisé le comportement des flux RSS en taux de publication et leur type, nous nous intéressons dans cette section à étudier la structure ainsi que la taille des items publiés par ces flux. Nous étudions ensuite la réplication des items dans les flux.

3.4.1 Structure des items

Les résultats présentés dans le tableau 3.6, montrent qu'un grand nombre de balises du format XML des flux RSS ou bien Atom ne sont pas utilisées dans les items. Bien que *title*, *description* et *link* sont présentes dans la majorité des items, *pubDate* n'est pas présente dans 20% des items, et la balise *language* n'apparaît pas dans 30% des flux. Presque 2/3 des items ne sont pas catégorisés (balise *category* absente) tandis que l'information concernant l'auteur (*author*) est présente dans moins de 8% des items. Il est à noter que 16% des balises contiennent des erreurs ou bien sont utilisées avec un faux format (mélange des balises RSS ou bien Atom). Il existe d'autres balises XML qui ne sont pas présentes dans notre jeu d'items, ne sont pas présentées dans le tableau 3.6. Pour conclure, les items RSS/Atom sont plus caractérisés par des balises qui présentent leur contenu textuel (*title* et *description*), que d'autres informations comme leur catégorie (*category*) et leur auteur (*author*).

title	**link**	**pubDate**	**desc.**	**Language**
99,82%	99,88%	80,01%	98,09%	69,14% (flux)
author	**category**	**GUID**	**ext.**	**RSS/Atom**
7,51%	33,94%	69,50%	29,73%	16,48%

Table 3.6 – Popularité des balises XML dans les items

3.4.2 Taille des items

Ensuite, nous nous sommes focalisés sur la taille des items. Comme nous avons pu le constater, malgré le format XML de RSS et Atom, le principal contenu renseigné (99,5%) dans un item est son contenu textuel (titre ou description). De fait, pour notre système d'indexation (proposé dans chapitre 4), il est nécessaire de connaître la taille des items en nombre de mots afin de mieux évaluer le modèle et la complexité de celui-ci, afin de définir précisement le type d'indexation appropriée.

La taille des items, mesurée en nombre de termes, correspond à l'extraction du titre et la description. Nous avons donc pu constater que les items contiennent en moyenne 52 termes correspondent à 36 termes distincts (voir tableau 3.7). Il apparaît que les items de flux RSS/Atom sont plus grands que des annonces publicitaires (4-5 termes [König et al., 2009]), des requêtes Web (2-4 mots [Zien et al., 2001, Spink et al., 2001])

ou des *tweets* (environ 15 termes [Oxford, 2009]), mais plus petits que les messages de blogs (250-300 termes [Ma and Zhang, 2007]) ou bien les pages Web (450-500 termes [Levering and Cutler, 2006]). Ces résultats montrent qu'il est nécessaire d'adapter un système d'indexation à ces tailles différents de ceux proposés pour la recherche d'information où de larges documents sont indexés. La variance élevée (11 885,90 pour la taille de l'ensemble de termes de l'item et 2 061,58 pour l'ensemble de termes distincts) est dûe à la large diversité de la balise description qui peut aller de l'absence de texte à un large texte (la totalité du fichier XML).

	Totalité des Termes			**Ensemble de Termes**		
	Titre	**Description**	**Total**	**Titre**	**Description**	**Total**
Moyenne	6,81	45,56	**52,37**	6,68	33,32	**36,71**
Max	235	10 256	10 262	144	2 369	2 370
Variance	12,97	11 821,36	11 885,90	11,36	2 081,75	2 061,58

Table 3.7 – Caractérisation du contenu textuel des items

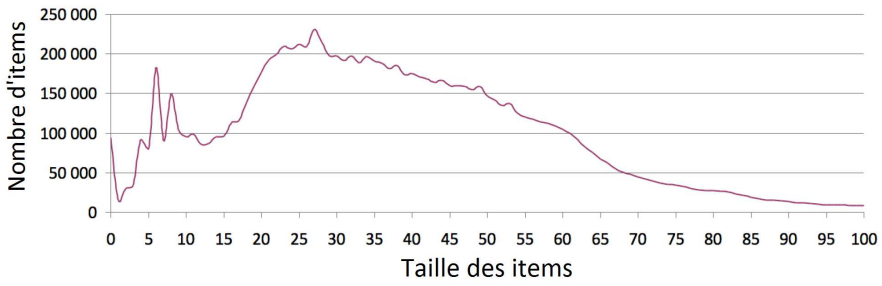

Figure 3.6 – Nombre d'items par taille d'item

La figure 3.6 nous montre la distribution du nombre d'items par rapport à leur taille en nombre de termes distincts. Nous pouvons observer une longue queue. Elle correspond à celle décrite dans la littérature pour la distribution de la taille des documents du Web [Williams and Zobel, 2005]. Nous pouvons constater que 51,39% des items ont une taille entre 21 et 50 termes, et 14% entre 8 et 20 termes. Les pics des tailles 6 et 8 sont dues aux items publiés par des flux de types commerciaux (produisent >55% des items de cettes tailles) dont leurs items respectent un modèle de taille fixe (*par ex.*, les items se diffèrent seulement par un ou bien quelques termes).

Dans la section 3.3.1, nous avons conclu que le comportement des flux en taux de publication était différent entre chaque type de flux. Ainsi, nous nous sommes intéressés à étudier la taille des items en nombre de termes distincts pour chaque type de flux. La figure 3.7 montre le pourcentage cumulatif du nombre d'items (fonction de répartition *cumulative distribution function*, notée par CDF) par rapport à la taille des items. Nous pouvons distinguer trois comportements principaux : (*i*) la majorité des items de `Press` (ligne continue) ont une taille entre 20 et 60 (80%) et seulement 10% ont moins de 10 termes ou bien plus de 60 termes ; (*ii*) les items des `Blogs` (ligne pointillée) ont des items

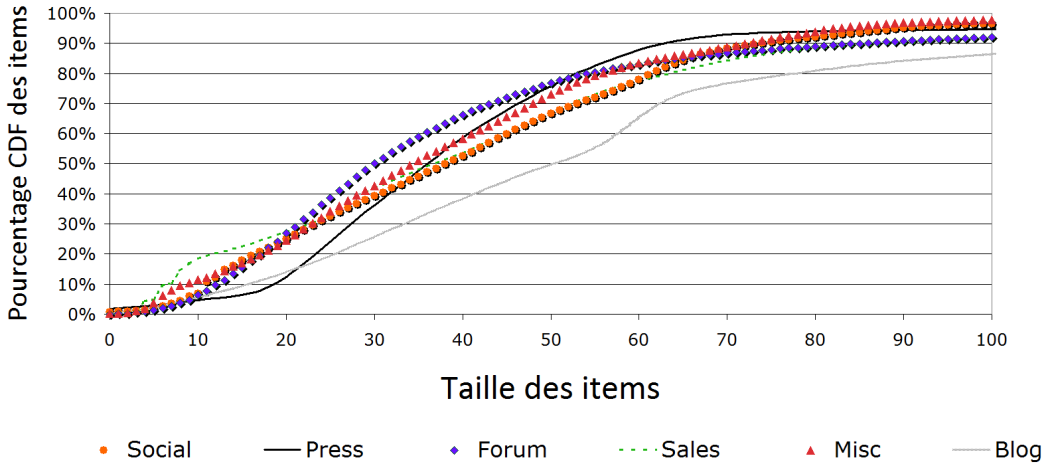

Figure 3.7 – Pourcentage CDF de la taille des items pour chaque type

plus larges que d'autres types de flux où 50% des items ont une taille supérieure à 50 termes (et 20% ont plus de 80 termes) ; la distribution de la taille des items de ce type est presque uniforme comme le présente la courbe cumulative qui est presque linéaire ; (iii) les items des autres types de flux ont de petites tailles, 30% inférieures à 20 termes et 75% moins de 50 termes.

[Downey, 2001] proposent une modélisation de la taille des items par une distribution log-normal en utilisant l'équation 3.3, où μ et σ sont respectivement la valeur logarithmique de la moyenne des tailles et la déviation standard, et Φ est une CDF normale standard. Cette distribution caractérise la taille de tous les items en prenant tous les types de flux en considération, avec $\mu = 3,59$ et $\sigma = 0,56$ (*Chi-square* = 0,42 pour 80 degrés de liberté). Alors nous concluons que la valeur moyenne de cette distribution est égale à $\exp(\mu) = 36,23$. Cette distribution présente une longue queue pour les items qui sont très larges (CDF près de 100%), mais ne prend pas en considération la distribution des petits items (items de `Sales` qui respectent un modèle pour leur structure).

$$F(x;\mu,\sigma) = \Phi\left(\frac{\ln(x)-\mu}{\sigma}\right) \tag{3.3}$$

Finalement, nous présentons dans le tableau 3.8 les tailles moyennes des items de chaque type de flux en nombre total de termes et en nombre de termes distincts. Nous observons que les flux des `Blogs` ont des items plus larges avec 72 termes que ceux des `Press` qui ont 61 termes. Cependant avec 48 et 40 termes distincts respectivement, les deux types ont le même comportement en réplication de termes avec un taux d'un tiers. Contrairement aux items de `Misc.` qui sont les plus petits avec seulement 40 termes (30 termes distincts).

La principale conclusion de l'analyse de la taille des items publiés par les flux RSS/Atom est que ces items sont plus larges, que les offres des publica-

Type	**Press**	**Misc.**	**Social**	**Sales**	**Forums**	**Blogs**
Totalité des termes	60,81	40,13	46,03	48,07	47,26	71,77
Ensembles de termes distincts	40,64	30,52	34,12	34,70	34,15	47,85

Table 3.8 – Taille des items par type

tions (*advertisement bids*) (4-5 termes [König et al., 2009]) ou bien les *tweets* (∼15 termes [Oxford, 2009]) mais plus petits que les messages originaux publiés sur les blogs (250-300 termes [Ma and Zhang, 2007]) ou bien les pages Web (450-500 termes sans les balises [Levering and Cutler, 2006]). Toutefois, la taille des items dépend de sa source d'information (type de flux) qui se reflète par des distributions différentes. En fait, les items de `Press` sont principalement entre 20 et 60, avec un petit résumé de l'information publiée et un lien vers un article très long, `Blogs` (où en général le contenu des items est des commentaires ou bien des opinions des utilisateurs) sont plus de 50 termes, et les items des `Sales` qui suivent de petits modèles pour leur structure, sont entre 5 et 30 termes.

3.4.3 Réplication des items

Une de nos études sur la structure des items s'est focalisée sur la réplication des items par les flux hébergés par le même site (réplication *intra-site*) et celle entre les différents sites (réplication *inter-site*). Pour détecter la réplication des items, nous avons utilisé une fonction de hachage du contenu des items (titre et description). Après avoir éliminé les items dupliqués par le même flux (réplication *intra-flux*), parmi les 27 millions items collectés nous avons extrait 10,7 millions items distincts (un item unique par flux).

% de réplication	$< 10\%$	10 - 19%	20 - 29%	30 - 90%	$\geq 90\%$
% de flux	68,23%	6,92%	4,19%	11,45%	9,21%
% d'items	23,53%	5,45%	5,18%	17,78%	48,06%

Table 3.9 – Réplication intra-flux : pourcentage des flux et d'items

Le tableau 3.9 présente le pourcentage de réplication dans un même flux (réplication *intra-flux*) pour les 27 millions items acquis. Pour chaque taux de réplication nous présentons le pourcentage de flux correspondant. Nous remarquons que la majorité des flux ne dupliquent pas leurs items. Dans la deuxième ligne du tableau, nous présentons le pourcentage des items publiés par ces flux, la majorité des items sont dupliqués par seulement 9% des flux. Parmi ces flux, 98% sont de la classe `Slow` (principalement des flux de `Misc.` et `Sales`), et seulement 0,9% (1,13%) de ces flux sont de la classe `Productive` (resp. `Moderate`).

Les sites Web publient leurs informations sur plusieurs flux, pour cela nous nous sommes intéressés à étudier la réplication des items dans les flux RSS provenant du même site :

% de réplication	**< 10%**	**10 - 19%**	**20 - 29%**	**≥ 30%**
% des sites	95,19%	1,09%	0,68%	3,04%
% d'items	71,31%	10,88%	10,23%	7,58%

Table 3.10 – Réplication intra-site : pourcentage des sites et d'items

réplication *intra-site* (même adresse *IP*) en comparant les items publiés par le même site. Nous avons rassemblé tous les sous-flux provenant du même site sous un seul site (*par ex.*, « rss.nytimes.com » et « blogs.nytimes.com » deviennent « nytimes.com »). Le tableau 3.10 montre que la réplication *intra-site* qui a été mesurée sur la période de 8 mois pour les 2 930 sites (différentes adresses *IP*) des 8 155 flux. 95% des sites (qui publient 70% des items) ont un pourcentage de réplication inférieur à 10%. Un petit nombre de sites (moins que 5%) de type `Sales`, `Press` et `Blogs` publient plus d'items répliqués (taux de réplication > 30%).

Une fois les réplications *intra-site* supprimées, nous avons mesuré la réplication *inter-site*. Le tableau 3.11 montre, pour chaque item dupliqué, le nombre de flux *distincts* dans lesquels il apparaît. Il apparaît clairement que la réplication entre les différents sites est négligeable : il présente moins que 0,5% du nombre total des items. Ce comportement est expliqué par l'absence dans notre jeu de données des flux RSS comme GoogleReader [6] ou Yahoo Pipes [7] qui agrègent les flux de différents sites.

# de flux distincts	**Une fois**	**Deux fois**	**≥ Trois fois**
% d'items	99,51%	0,41%	0,08%

Table 3.11 – Pourcentage d'items dupliqués par plusieurs sites

Puis, le taux de réplication par item est donné dans le tableau 3.12 qui présente pour chaque type de flux le pourcentage d'items dupliqués et le nombre de fois. En général, les items ne sont jamais dupliqués mais le taux de duplication varie d'un type à un autre. Nous remarquons que `Misc.`, `Forum` et `Press` ont un taux de duplication plus élevé pour une double publication de l'item, tandis que `Sales`, `Misc` et `Press` possèdent un taux de publication multiples non négligeable. Cela correspond à un comportement de réplication spécifique au sein de ces types qui diffusent l'information plus largement que les autres types.

Plusieurs métriques basées sur des fonctions de similarité ont été proposées dans la littérature pour la détection de réplication [Elmagarmid et al., 2007]. Les métriques *Jaccard* et *Jaro-Winkler* sont des métriques très pertinentes pour la mesure de réplication dans les flux RSS. La mesure *Jaccard* évalue le nombre de termes en commun sur le nombre total de termes entre deux ensembles de termes X et Y (équation 3.4). Tandis que *Jaro-Winkler* calcule la réplication entre deux chaînes de caractères s et t en se basant sur le nombre de caractères en commun (s' et t') et le nombre de transpositions nécessaires

6. www.google.com/reader
7. pipes.yahoo.com

Type	Une fois	Deux fois	Trois fois	≥ Quatre fois
Social	89,34%	7,76%	2,87%	0,03%
Press	83,94%	10,62%	3,67%	1,77%
Forums	88,51%	11,47%	0,02%	0,00%
Sales	86,42%	9,64%	1,46%	2,48%
Blogs	88,79%	8,01%	2,41%	0,79%
Misc.	79,55%	10,65%	7,29%	2,52%

Table 3.12 – Réplication vs types : % des items répliqués X fois

($T_{s',t'}$) pour les faire correspondre (métrique *Jaro* donnée par l'équation 3.5), donnant plus de poids quand les deux chaînes ont un préfixe commun par la métrique *Jaro-Winkler* (équation 3.6) ; cette spécificité minimise l'effet des fautes d'orthographe. Nous avons utilisé ces métriques pour mesurer la similarité entre les titres et les descriptions des items *intra-flux*. Pour calculer la similarité, nous avons utilisé la librairie *Simmetrics*[8].

$$jaccard = \frac{|X \wedge Y|}{|X \vee Y|} \tag{3.4}$$

$$jaro = \frac{1}{3}\left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{2 \times |s'|}\right) \tag{3.5}$$

$$jaro_{winkler} = jaro(s,t) + (prefixLength \times scale \times (1 - jaro(s,t))) \tag{3.6}$$

Nous présentons dans le tableau 3.13 le pourcentage de flux par distance *Jaccard* pour le titre et la description. Nous remarquons que la majorité des flux ne sont pas similaires (89% des flux ne partagent pas plus de 10% de l'information entre leurs items), et 99% des flux ne partagent pas la moitié des termes entre leurs items. Nous observons tout de même une petite différence pour la description où les items des flux ont tendance à être plus similaires.

Valeur moyenne	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	≥ 0,9
Titre	88,35	7,21	1,96	1,45	0,57	0,32	0,10	0,03	0,01
Description	89,11	5,14	2,35	1,56	0,83	0,58	0,28	0,05	0,10

Table 3.13 – Pourcentage de similarité *intra-flux* pour la mesure *Jaccard*

Le tableau 3.14 donne le pourcentage du nombre de flux pour une valeur moyenne en utilisant la métrique *Jaro-Winkler* pour le titre et la description. Nous observons que la majorité des flux ont des titres très différents (beaucoup de transpositions), 86% des flux

8. IntelliJ IDEA : www.dcs.shef.ac.uk/∼sam/simmetrics.html

ont une valeur de *Jaro-Winkler* > 0,5. Un petit nombre de flux (moins de 7%) possèdent une valeur élevée de *Jaro-Winkler* (> 0,7) pour leurs titres ce qui est très différents des descriptions où 47% des flux ont une valeur > 0,7, c'est-à-dire qu'elles ont beaucoup de caractères en commun surtout en début de chaînes de caractères. Cela est dû à la présence de modèles (*patterns*) et de structures très répétés (texte exact et/ou structure HTML) dans la description des items de certains flux.

Valeur moyenne	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	≥ 0,9
Titre	2,33	2,11	8,36	40,00	33,52	7,13	2,76	2,31	1,48
Description	5,65	1,10	1,75	4,65	13,62	26,24	24,73	14,18	8,09

Table 3.14 – Pourcentage de la similarité *intra-flux* pour la mesure *Jaro-Winkler*

Avec une distance *Jaccard* globale moyenne (resp. *Jaro-Winkler*) égale à 0,081 (resp. 0,459) pour la similarité des titres et 0,109 (resp. 0,625) pour les descriptions, les flux ne dupliquent pas l'information déjà publiée, mais ils dupliquent plus régulièrement quelques informations dans leurs descriptions. En se basant sur cette conclusion, nous nous sommes intéressés à étudier ce comportement de duplication dans les descriptions en fonction de type du flux[9]. Le tableau 3.15 montre pour chaque type son pourcentage en nombre de flux qui ont des valeurs de *Jaccard* et *Jaro-Winkler* supérieures à 0,5. Nous observons que ces pourcentages sont très similaires pour les `Blogs`, `Misc.` et `Sales`. La majorité des réplications en termes se situe dans les flux de type `Sales` (47%) et `Misc.` (27%), mais la réplication en modèle et structure (*patterns*) est en majorité dans les flux de type `Misc.` (40%) et `Blogs` (35%). Par contre, les flux de type `Social Media` et `Forums` possèdent un petit pourcentage de réplication *intra-flux* (les deux représentent 3% des flux qui ont des « descriptions similaires »).

Métrique	**Blogs**	**Forums**	**Misc.**	**Press**	**Sales**	**Social Media**
Jaccard	20,41%	0,68%	27,21%	2,04%	47,62%	2,04%
Jaro-Winkler	35,62%	0,93%	40,04%	10,39%	11,28%	1,75%

Table 3.15 – Réplication *intra-flux* élevée dans la description *% aux* types des flux

Pour conclure, nous avons observé que la re-publication des items entre les flux est assez faible où nous avons trouvé que seulement 0,41% des items sont dupliqués par des flux de différents sites. Il est à noter que *99%* des flux n'ont pas un taux de similarité très élevé (similarité *Jaccard* ≤ 0,5), c'est-à-dire que les items ne partagent pas sur un même flux plus de la moitié de leurs termes. Mais 47% des flux (particulièrement les flux de `Misc.`, `Blogs` et `Sales`) ont des descriptions très similaires car ils respectent des modèles ou bien structures pour publier leurs informations, cela indique que l'information principale publiée par les flux RSS est présentée dans le titre des items.

9. Exemples des flux très dupliqués par similarité :
rss.groups.yahoo.com/group/curriculumevaluation/rss
www.freebookzone.com/rss/genrss.php ?cmd=pop&format=rss2_0

3.5 Analyses du vocabulaire

Nous nous sommes ensuite intéressés à l'étude du contenu textuel des items. Pour limiter les effets de bords du langage, nous avons concentré notre jeu de données sur les items provenant des flux `Anglais`. Le vocabulaire ainsi obtenu atteint 1 537 730 termes parmi lesquels seulement 4% font parti du dictionnaire *WordNet* [Miller, 1995]. Cela est dû à la présence d'un grand nombre de noms propres (noms de personnes et de lieux), ainsi que d'un grand nombre de fautes d'orthographe et d'abréviations. À partir de ces constatations, nous avons distingué deux vocabulaires distincts : $V_{\mathcal{W}}$, le vocabulaire de (61 845) termes qui apparaissent dans *WordNet* et $V_{\overline{\mathcal{W}}}$, le vocabulaire de (1 475 885) termes restants. Cela nous permettra d'identifier des éventuels effets indésirables sur le vocabulaire global. Les *Stop-words*[10], les URLs et les adresses mél ont été supprimés. Pour $V_{\overline{\mathcal{W}}}$, le terme est ajouté sous sa forme racine (*stemmed*) et pour $V_{\mathcal{W}}$ le terme est ajouté sous la forme telle qu'elle apparaît dans *WordNet*.

3.5.1 Occurrences des termes

La caractérisation de la distribution du vocabulaire nous permet de mieux appréhender la répartition des rangs des termes du vocabulaire. Ainsi, la figure 3.8 représente le nombre d'occurrences des termes de $V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$ dans l'ordre décroissant de leur rang (fréquence) dans leur vocabulaire respectif. Nous constatons que les termes de $V_{\mathcal{W}}$ sont plus fréquents que les termes de $V_{\overline{\mathcal{W}}}$. La distribution des occurrences des termes a été caractérisée par une distribution *stretched exponential*. En effet, contrairement aux distributions trouvées dans la littérature (*loi Zipf* [Baeza-Yates and Ribeiro-Neto, 1999, Manning et al., 2008]), la *stretched exponential* prend mieux en compte la longue queue constatée pour les termes les moins fréquents. D'autre part, il est intéressant de noter que la somme des occurrences des 5 000 termes les plus fréquents de $V_{\mathcal{W}}$ représente 87% des occurrences total du vocabulaire.

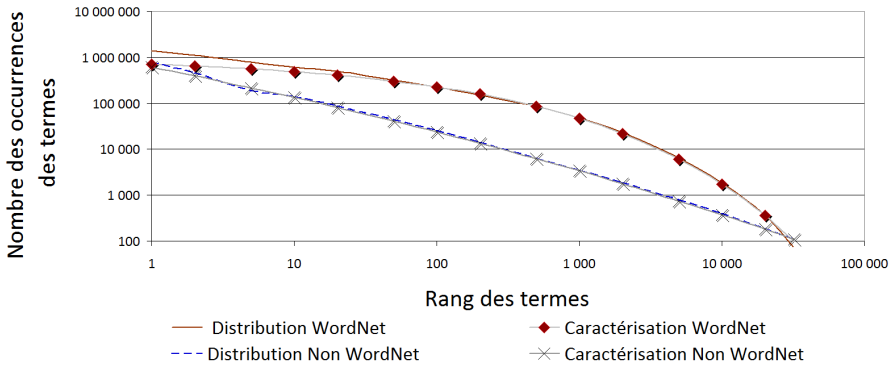

Figure 3.8 – Distribution des occurrences des termes du $V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$

10. www.lextek.com/manuals/onix/stopwords2.html

Par la suite, nous caractérisons formellement la distribution des occurrences des termes pour les deux vocabulaires $V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$. Dans ce but, la distribution de la loi *Zipf* (équation 3.7) a été utilisée dans la littérature [Baeza-Yates and Ribeiro-Neto, 1999, Manning et al., 2008] pour différents corpus de textes :

$$f(r) = \frac{K}{r^{\theta}} \tag{3.7}$$

où r est le rang du terme et θ et K sont des constantes.

Cependant, comme le montre la figure 3.8 la courbe qui correspond à la distribution de $V_{\mathcal{W}}$ possède une déviation significative de la loi *Zipf*, *c.-à-d.,* ne correspond par à une ligne droite dans une échelle log-log. Cette déviation est plus petite pour la distribution de $V_{\overline{\mathcal{W}}}$. Ce comportement de la distribution (déviation de la ligne droite) a été aussi rapporté dans la littérature pour les textes provenant du Web [Manning et al., 2008, Baeza-Yates and Ribeiro-Neto, 1999, Ahmad and Kondrak, 2005, Dhillon et al., 2001]. D'autres études statistiques sur les documents du Web [French, 2002], requêtes sur le Web [Spink et al., 2001, Williams and Zobel, 2005, Jansen et al., 1998, König et al., 2009], blogs [Lambiotte et al., 2007] et offres des publications sur le Web [König et al., 2009] observent également cette déviation pour leurs résultats, et peu de travaux ont essayé de trouver des distributions plus adéquates. [Montemurro, 2001] essaye de généraliser la loi *Zipf* en proposant la distribution *Zipf - Mandelbrot* (équation 3.8) tandis que [Lambiotte et al., 2007] propose d'utiliser une distribution *Power Law* modifiée (équation 3.9).

$$f(r) = \frac{K}{(c+r)^{\theta}} \tag{3.8}$$

où K, c et θ sont des constantes.

$$f(r) = \frac{1}{1 + \alpha 1 \times r^{\gamma 1} + \alpha 2 \times r^{\gamma 2}} \tag{3.9}$$

où $\alpha 1$, $\alpha 2$, $\gamma 1$, $\gamma 2$ sont des constantes.

Les lois *Zipf - Mandelbrot* et *Power Law* modifiée permettent de caractériser le début des courbes de la distribution des occurrences des termes. Mais elles ont une longue queue qui est une ligne droite dans une échelle en log-log, ce qui ne correspond pas aux distributions des occurrences de nos vocabulaires, ni aux distributions étudiées dans la littérature.

Nous avons trouvé la distribution *Stretched Exponential* (équation 3.10) qui permet de mieux caractériser la queue des courbes des distributions des occurrences de nos vocabulaires.

$$f(r) = K \times e^{-(r/c)^{\theta}} \tag{3.10}$$

où θ correspond à la pente de la courbe, et c une constante pour exprimer sa déviation : pour une valeur égale à zéro la courbe correspond à une ligne droite dans une échelle log-log. Cette distribution a été utilisée pour caractériser des phénomènes physiques mais jamais pour la caractérisation des distributions des termes, même si [Laherrère and Sornette, 1998] propose de l'utiliser comme une alternative pour la distribution *Power Law*. Les valeurs de test de *Chi-square* pour les différentes lois de distribution appliquées à nos courbes sont présentées dans le tableau 3.16. De petites valeurs de *Chi-square* indique que la loi est plus adaptée pour caractériser la distribution. Par conséquent, nous avons utilisé *Stretched Exponential* pour caractériser la distribution des occurrences des termes de nos vocabulaires (voir figure 3.8) puisqu'elle donne les plus petites valeurs avec le test *Chi-square*.

	$V_{\mathcal{W}}$	$V_{\overline{\mathcal{W}}}$	co-occurrences des termes
Zipf	2 845	11,5	2 353
Zipf - Mandelbrot	2 493	11,5	1 543
Power law modifiée	966	9,2	31 653
Stretched Exponential	49,8	9	45
Degré de liberté du test Chi-square	210	200	190

Table 3.16 – Valeurs *Chi-square* pour caractériser les distributions

La figure 3.9 présente les occurrences de 32 000 combinaisons de termes les plus fréquentes dans les items. Nous avons considéré les combinaisons de paire des termes, qui ne sont pas nécessairement consécutives, qui apparaît dans le titre ou la description de l'item quelque soit le vocabulaire auquel ils appartiennent. Dans [König et al., 2009], ils ont caractérisé la distribution des combinaisons pour des données de *sponsored search* par la loi *Zipf*, avec laquelle nous n'avons pas pu caractériser notre distribution. En effet, la figure 3.9 présente la courbe des combinaisons des termes que nous avons aussi caractérisé par une distribution *Stretched Exponential*, car elle permet également d'obtenir les plus petites valeurs du test *Chi-square* (voir tableau 3.16).

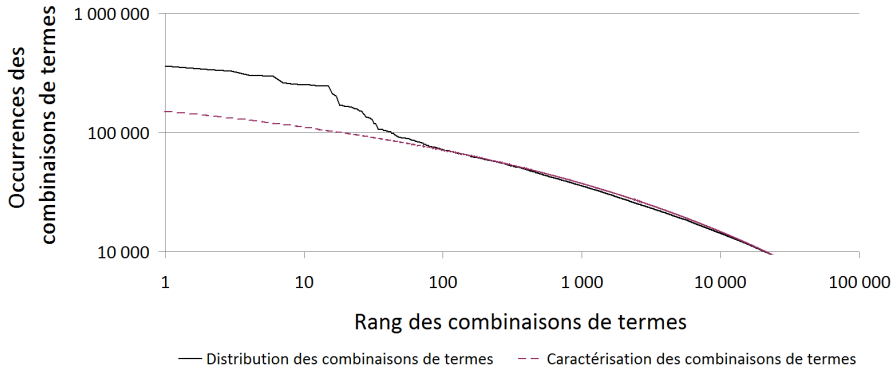

Figure 3.9 – Distribution des occurrences des combinaisons de termes

Nous observons un écart entre la courbe des combinaisons des termes et sa représentation avec la distribution *Stretched Exponential* pour les 32 combinaisons les plus fréquentes qui apparaissent presque 4 fois plus que ce qui est attendu. Ce comportement est également observé dans [König et al., 2009].

Le tableau 3.17 présente les constantes utilisées pour caractériser les distributions des occurrences des termes selon leur rang dans les deux vocabulaires ($V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$) et celle des co-occurrences des termes par la loi *Stretched Exponantial* (figures 3.8 et 3.9).

	$\lvert V_{\mathcal{W}}\rvert$	$\lvert V_{\overline{\mathcal{W}}}\rvert$	co-occurrence des termes
K	970 000	1 200 000	450 000
c	31,6	$6{,}92 \times 10^{-21}$	0,9
θ	0,32	0,053	0,133

Table 3.17 – Les constantes de *Stretched Exponential*

Le tableau 3.18 montre la proportion de termes de $V_{\mathcal{W}}$ parmi les termes les plus fréquents dans le vocabulaire global (termes dans $V_{\mathcal{W}} \cup V_{\overline{\mathcal{W}}}$). Nous observons que les termes du $V_{\mathcal{W}}$ présentent un pourcentage important parmi les termes les plus fréquents, plus de 90% des 5 000 premiers termes appartiennent à $V_{\mathcal{W}}$. Ce résultat est prévisible comme les termes du $V_{\mathcal{W}}$ sont des termes communs utilisés dans les conversations et la littérature. En plus, 78% des termes de rang entre 5 000 et 10 000 préviennent de $V_{\mathcal{W}}$, mais ce pourcentage diminue pour les autres tranches des rangs termes ; 60% pour les rangs entre 10 000 et 20 000, et seulement 3% du reste des termes.

Rangs des termes	1-5000	5001-10000	10001-15000	15001-20000	$>$ 20000
Proportion $V_{\mathcal{W}}$	0,91	0,78	0,67	0,58	0,03

Table 3.18 – Proportion des termes de $V_{\mathcal{W}}$ parmi les termes les plus fréquents

Nous remarquons dans la figure 3.8 que le nombre d'occurrences des termes se diffère d'un terme à un autre. Nous présentons dans le tableau 3.19 le nombre de termes pour des occurrences faibles. Les termes dont le nombre d'occurrences est inférieur à 5, présentent 76% du vocabulaire global tandis que les termes qui apparaîssent une seule fois représentent 45% du vocabulaire.

Occurrences	1	2	3	4	≥ 5
# termes	659 159	261 906	124 450	77 814	414 401

Table 3.19 – Nombre de termes par occurrence

3.5.2 Analyse des termes

Ensuite, nous avons analysé la nature d'un ensemble de 5 000 termes parmi les termes de $V_{\overline{\mathcal{W}}}$ qui n'ont qu'une seule occurrence. Ces termes ont été choisis aléatoirement parmi

les termes qui n'apparaissent qu'une seule fois, puis ils ont été manuellement analysés. La figure 3.10 montre la répartition de ces types de termes. D'après notre analyse, la majorité de ces termes (35%) sont des noms d'entités (noms propres et lieux) tandis que les fautes d'orthographe ne présentent que 5% de ces termes. Bien que nous n'ayons utilisé que les items des flux `Anglais` pour extraire notre vocabulaire, nous avons trouvé que 22% des termes appartenaient à d'autres langues. À noter aussi la présence d'autres types de termes comme de fausses URLs, noms composés, caractères spéciaux et de nombreux mots techniques...

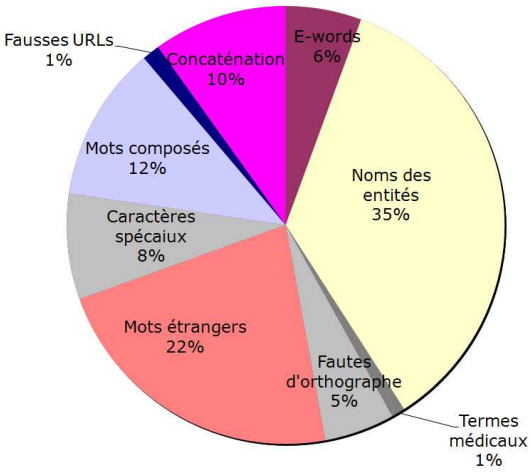

Figure 3.10 – Nature des termes ayant une seule occurrence

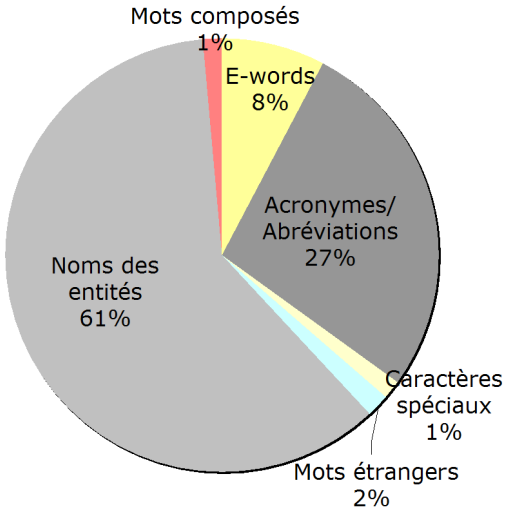

Figure 3.11 – Nature des termes les plus fréquents de $V_{\overline{W}}$

La figure 3.11 affiche les types des 5 000 termes les plus fréquents de $V_{\overline{W}}$ (dont le nombre d'occurrences est plus grand que 10^4). Nous observons que les noms des entités (noms propres et lieux) sont les plus fréquents (61%), puis les acronymes et les abréviations (27%). Ainsi que la présence de nouveaux mots de l'Internet (E-words) (8%).

Il est connu que le langage utilisé sur le Web est soumis à de nombreuses erreurs et fautes d'orthographe [Ahmad and Kondrak, 2005], ainsi que l'utilisation des acronymes et les abréviations. Ces imperfections sont dues à l'absence d'un langage formel sur le Web 2.0. Bien que cette analyse soit faite sur un ensemble choisi aléatoirement de termes qui n'ont qu'une seule occurrence, et sur l'ensemble des termes les plus fréquents, nous pouvons dire que les noms des entités (noms propres et lieux) présentent un pourcentage important du vocabulaire des termes qui n'apparaissent pas dans le dictionnaire ($V_{\overline{W}}$), mais il est à noter que les termes techniques utilisés sur le Web (*E-words*) et les abréviations occupent une place non négligeable dans ce vocabulaire.

3.5.3 Evolution de la taille du vocabulaire

Afin de prévoir l'évolution de la taille du vocabulaire au cours du temps, nous avons mesuré celle-ci en fonction de l'arrivée séquentielle des items de notre jeu de données. La figure 3.12 illustre cette évolution pour les deux vocabulaires. Nous constatons que la taille de $V_{\mathcal{W}}$ évolue plus rapidement que celle de $V_{\overline{\mathcal{W}}}$. Toutefois, à la fin des 8 mois d'acquisition des données, $V_{\overline{\mathcal{W}}}$ est 24 fois plus grand que $V_{\mathcal{W}}$ ($|V_{\mathcal{W}}| = 61\ 845$ tandis que $|V_{\overline{\mathcal{W}}}| = 1\ 475\ 885$). Une évolution rapide de $V_{\mathcal{W}}$ est observée pour les 140 000 premiers items pour atteindre les 31 000 termes les plus fréquents de $V_{\mathcal{W}}$. La croissance est bornée par la taille du vocabulaire anglais (*WordNet*). D'un autre côté, $V_{\overline{\mathcal{W}}}$ augmente plus lentement. En effet, les 140 000 premiers items de $V_{\overline{\mathcal{W}}}$ ne génèrent qu'un vocabulaire de 88 000 termes, c'est-à-dire seulement 6% de sa taille totale.

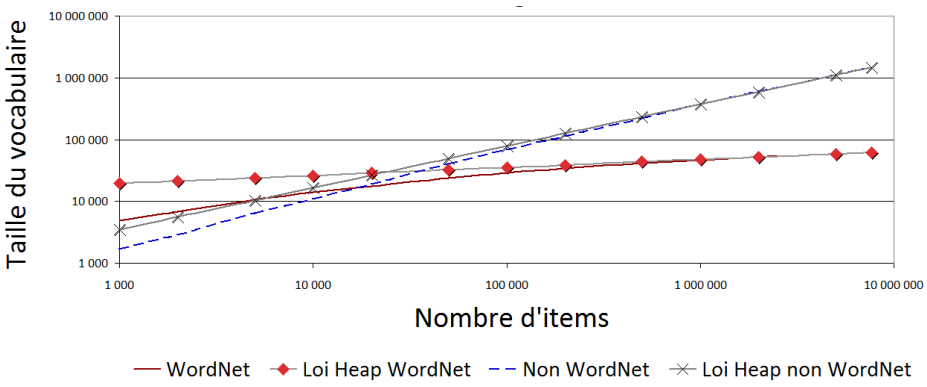

FIGURE 3.12 – Evolution de la taille des vocabulaires

L'évolution de la taille du vocabulaire est généralement caractérisée dans la littérature par la distribution de la loi de *Heap* [Baeza-Yates and Ribeiro-Neto, 1999, Manning et al., 2008] (voir figure 3.12), définie comme suit :

$$|V(n)| = K \times n^{\beta}$$

où n est le nombre d'items, tandis que K et β ($\in [0,1]$) sont des constantes qui dépendent du jeu de données à caractériser et de la méthode utilisée pour extraire le vocabulaire [Williams and Zobel, 2005]. β détermine la vitesse de l'évolution du vocabulaire avec le temps, avec des valeurs entre 0,4 et 0,6 pour des collections de texte homogènes et de taille moyenne [Baeza-Yates and Ribeiro-Neto, 1999]. La valeur utilisée dans [Williams and Zobel, 2005] pour la loi de *Heap* afin de caractériser l'évolution du vocabulaire d'une collection de 500 Mo de documents du journal *Wall Street* est différente ($\beta = 0,16$) ; à noter que les mots de cette collection sont des séquences alphanumériques. Nous présentons dans le tableau 3.20 les constantes de la loi de Heap pour caractériser l'évolution de nos vocabulaires $V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$ ainsi que le vocabulaire global ($|V_{\mathcal{W}} + V_{\overline{\mathcal{W}}}|$).

Il apparaît clairement que l'évolution du vocabulaire global (présentée par β) est plus affectée par $V_{\overline{\mathcal{W}}}$ que $V_{\mathcal{W}}$, cela est dû à sa petite taille par rapport à celle de $V_{\overline{\mathcal{W}}}$.

	$\lvert V_{\mathcal{W}} + V_{\overline{\mathcal{W}}}\rvert$	$\lvert V_{\mathcal{W}}\rvert$	$\lvert V_{\overline{\mathcal{W}}}\rvert$
K	51	7 921	33
β	0,65	0,13	0,675

Table 3.20 – Constantes de la loi de *Heap*

L'exposant de $V_{\overline{\mathcal{W}}}$ (0,675) est un peu plus élevé que ceux présentés dans la littérature [Baeza-Yates and Ribeiro-Neto, 1999, Manning et al., 2008] ce qui montre l'évolution rapide du vocabulaire de nos items. Il est à noter que ce comportement est également présent pour d'autres types de vocabulaires comme ceux des requêtes textuelles générées par les utilisateurs (comme les requêtes sur le Web [Zien et al., 2001, Schmidt-Maenz and Koch, 2005]). Finalement, le cœfficient élevé de la loi de *Heap* (K) pour $V_{\mathcal{W}}$ est expliqué par l'évolution rapide du vocabulaire au début de l'acquisition des termes les plus utilisés.

Type	$V_{\mathcal{W}} + V_{\overline{\mathcal{W}}}$	$V_{\mathcal{W}}$	$V_{\overline{\mathcal{W}}}$
Social Media	$172 \times n^{0,55}$	$2426 \times n^{0,22}$	$26 \times n^{0,68}$
Press	$255 \times n^{0,53}$	$3731 \times n^{0,18}$	$135 \times n^{0,57}$
Forums	$53 \times n^{0,65}$	$6094 \times n^{0,13}$	$20 \times n^{0,71}$
Sales	$111 \times n^{0,56}$	$1061 \times n^{0,26}$	$51 \times n^{0,59}$
Misc.	$12 \times n^{0,74}$	$974 \times n^{0,28}$	$7 \times n^{0,77}$
Blogs	$54 \times n^{0,66}$	$7329 \times n^{0,14}$	$12 \times n^{0,77}$

Table 3.21 – Constantes de la loi de *Heap* pour $V_{\mathcal{W}}$ et $V_{\overline{\mathcal{W}}}$ pour chaque type de flux

Le tableau 3.21 présente pour chaque type de flux les constantes de la loi de *Heap* correspondant à leurs évolutions respectives. Nous observons que les flux de type `Press` possèdent le plus petit exposant ($\beta = 0{,}57$) pour $V_{\overline{\mathcal{W}}}$ et une valeur moyenne (0,18) pour $V_{\mathcal{W}}$, avec une valeur élevée pour K pour ses vocabulaires. Cette valeur élevée de K montre que le vocabulaire des items est grand et évolue rapidement, et qu'il comprend des termes de *WordNet* avec des noms propres et noms des lieux. Les journaux ne contiennent pas de fautes d'orthographe, ni des abréviations ce qui explique l'évolution faible de $V_{\overline{\mathcal{W}}}$ par rapport aux autres types de flux. Au contraire, les flux de type `Sales` possèdent une valeur élevée de β (0,26) pour $V_{\mathcal{W}}$, une petite valeur de K, et un petit exposant pour $V_{\overline{\mathcal{W}}}$ (0,59).

Il est à noter que les items du même type de flux partagent seulement un petit sous-ensemble de termes de $V_{\mathcal{W}}$, par exemple les items de type `Sales` qui ont un modèle standard, ainsi que de nouveaux termes apparaissent régulièrement avec l'arrivée de nouveaux items. L'acquisition des termes de $V_{\overline{\mathcal{W}}}$ est plus lente que celles des autres types ; les items de type `Sales` ne contiennent pas de noms d'entités. De plus, les items de `Sales` sont plus petits que les items des autres types (voir Table 3.8), ce qui explique l'évolution lente de son vocabulaire. Les flux des `Blogs` possèdent un autre comportement remarquable ; avec un petit exposant (0,14) pour $V_{\mathcal{W}}$ avec une valeur élevée pour K, et

l'exposant le plus élevé (0,77) pour $V_{\overline{\mathcal{W}}}$. La raison est que les blogs ne partagent qu'un petit sous-ensemble de termes et n'utilisent pas beaucoup de nouveaux termes de $V_{\mathcal{W}}$, d'où un vocabulaire commun très petit. Généralement, les blogs contiennent beaucoup de noms d'entités (noms propres), et aussi quelques fautes d'orthographe, acronymes et des abréviations..., ce qui explique l'acquisition rapide de son $V_{\overline{\mathcal{W}}}$.

3.5.4 Variation des rangs des termes

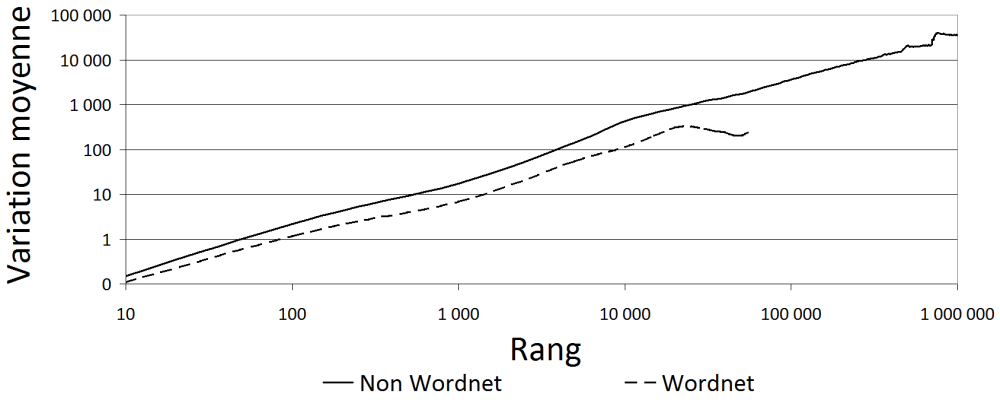

Figure 3.13 – Distance cumulative des rangs des termes

Pour finir, nous nous sommes intéressés à la variation des rangs des termes au cours de la période, ce qui nous permettra de mieux comprendre l'impact des mises-à-jour sur un index basé sur les rangs des termes. Nous avons étudié la variation moyenne du rang des termes du vocabulaire durant le dernier mois de la période d'acquisition de notre jeu de données. Cette variation est mesurée par la différence entre le rang du terme entre une semaine $t-1$ et une semaine t. Nous constatons dans la figure 3.13 que la variation est proportionnelle au rang, $D = \alpha \times r$, c'est-à-dire que les termes les moins fréquents sont sensés plus changer de rang par rapport aux termes du vocabulaire des semaines précédentes. Ce comportement peut avoir quelques conséquences sur les performances des index basés sur l'ordre des termes. La pente α de la variation de la distance D après huit mois est égale à 0,035 pour $V_{\overline{\mathcal{W}}}$ et 0,013 pour $V_{\mathcal{W}}$.

Nous avons également étudié l'évolution de cette variation pour des différentes classes de rangs pour $V_{\overline{\mathcal{W}}}$ d'une semaine à l'autre, mesurée sur les quatre dernières semaines de la période d'acquisition des items. Pour cela nous avons utilisé la distance *Spearman footrule* [Spearman, 1904, Kumar and Vassilvitskii, 2010]. Elle correspond à la somme des variations du rang d'un terme d'une semaine sur l'autre :

$$S(t) = \sum_{i=1}^{n} |r(t)_i - r(t-1)_i| \tag{3.11}$$

où t correspond à une semaine, $r(t)_i$ et $r(t-1)_i$ sont respectivement les rangs du terme i pendant la semaine t et $t-1$. La figure 3.14 montre les valeurs de *Spearman* pour trois classes de rangs de termes : (a) termes très fréquents (rangs entre 1 et 250), (b) termes couramment utilisés (10 000 à 10 250) ; et (c) termes rarement utilisés (500 000 à 500 250). À noter que ces termes sont choisis dans le vocabulaire acquis pendant les trois premiers mois de la période d'acquisition des items.

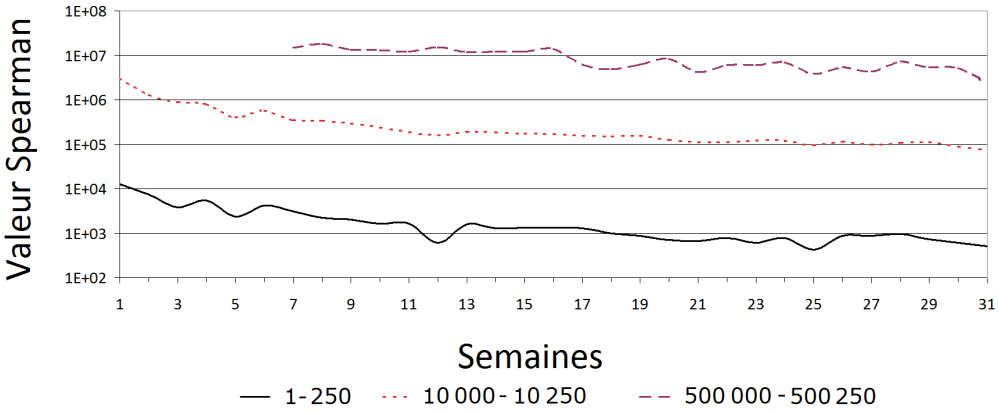

Figure 3.14 – Variation *Spearman* par semaine pour 3 classes de rangs de termes de $V_{\overline{\mathcal{W}}}$

Nous n'avons pas remarqué d'évolutions significatives des rangs des termes les plus fréquents durant toute la période d'acquisition. En fait, la somme des variations des rangs des termes les très fréquents est quatre ordres de magnitude plus petite que ceux rarement utilisés. Nous remarquons que la valeur de *Spearman* se stabilise après 20 semaines pour les trois classes des rangs. Cette stabilité peut être justifiée par le fait que le nombre d'occurrences de ces termes après cette période diminue l'impact de l'arrivée de nouveaux termes sur leur rang.

3.6 Conclusion

Nous avons effectué une étude sur le comportement des flux pour mettre en valeur certaines caractéristiques qui pourront nous être utiles lors de la modélisation des structures d'indexation Pub/Sub. Nous pensons que les résultats [Hmedeh et al., 2011a, Hmedeh et al., 2011b] de cette étude pourront également être utile dans différentes applications Web 2.0 comme : (i) la classification des flux selon leur activité [Beitzel et al., 2004], (ii) l'indexation de flux RSS par leur contenu [Hmedeh et al., 2012b, Hmedeh et al., 2012a, Hmedeh et al., 2013] ou bien par la similarité [McCreadie et al., 2010], (iii) l'amélioration du contenu textuel des items RSS grâce à la qualité du vocabulaire des termes et des fautes d'orthographe [Qingcheng and Youmeng, 2008], (iv) le classement ou la détection des spams

[Fotiou et al., 2010] en ce qui concerne la taille du vocabulaire et la distribution des occurrences des termes, (v) la prise en compte de l'évolution des rangs des termes pour des mises-à-jour d'index [Cambazoglu et al., 2010].

De plus, les réultats de notre analyse seront en particulier exploités dans la section 4.5 et la section 5.5 pour la validation de notre système de Pub/Sub pour la syndication Web en prenant en considération le comportement du Web. Le vocabulaire extrait (section 3.5) sera utilisé pour la génération de nos jeux de données (souscriptions) tout en respectant la distribution réelle de leurs occurrences. La validation du notre modèle analytique probabiliste proposé dans la section 4.3 est également basée sur les probabilités des occurrences des termes dans le vocabulaire extrait. Le jeu d'items réels est ainsi utilisé dans toutes nos expériences pour étudier les performances de nos index.

Nous allons maintenant nous intéresser à la conception d'index de type Publication/Souscription dont les modèles se baseront en parti sur l'étude que nous venons de présenter.

Chapitre 4

Indexation des souscriptions

4.1 Introduction

Dans les systèmes de Publication/Souscription du Web, le nombre de souscriptions est considérable, de l'ordre de plusieurs millions. De plus, le débit d'arrivée des items peut atteindre lors de certains événements des milliers d'items par seconde. Les items et les souscriptions sont des ensembles de mots-clés. Une souscription est satisfaite par un item si et seulement si l'ensemble de ses mots est inclus dans l'ensemble des mots-clés de l'item (sémantique du *broad match*). Dans ce contexte, nous nous intéressons dans ce chapitre à la conception d'un système de notification rapide et efficace. Afin de faciliter la notification des souscriptions et d'éviter les accès disques coûteux, les souscriptions doivent être indexées en mémoire centrale. L'intérêt de notre approche est de trouver la structure de données adéquate qui soit compacte en mémoire et efficace en recherche.

Dans cette thèse, nous nous sommes intéressés à une implantation efficace pour trois structures d'indexation. Les deux premières sont basées sur les *Listes Inverses* [Zobel and Moffat, 2006] et une troisième qui est une variante de l'*Ordered Trie* [Knuth, 1973] avec des termes distincts. Mais aussi à l'étude de leur comportement pour des paramètres critiques réalistes dans le cadre de la syndication web. Bien que ces structures de données aient été utilisées dans le contexte de la diffusion de l'information sélective [Yan and Garcia-Molina, 1994], les recherches sponsorisées [König et al., 2009] (*sponsored search*), ou l'exploitation d'ensembles fréquents d'items [Bodon, 2004, Malik and Kender, 2007], l'espace mémoire et le temps de traitement sont différents dans notre contexte. Cela est dû à la particularité des systèmes de la syndication web qui sont caractérisés [Hmedeh et al., 2011b] (a) par une taille d'items (25-36 termes distincts) plus grande que celle des publicités (*advertisement bids*) ; (4-5 termes) [König et al., 2009]) et plus petite que celle des documents du web (12K termes [Yan and Garcia-Molina, 1994]) (b) par un large vocabulaire (jusqu'à 1,5M termes). À noter aussi que dû à la sémantique du *broad match*,

les techniques de recherche d'information pour l'optimisation des listes inverses (c.-à-d., élimination rapide [Zobel and Moffat, 2006]) ne sont pas adaptées à notre contexte. Aucune étude détaillée de l'*Ordered Trie* n'a vraiment été réalisée. Toutefois l'usage de l'*Ordered Trie* dans les systèmes Pub/Sub n'a pas été recommandé à cause de leurs performances étudiées dans des travaux connexes (cf. filtrage de documents dans [Yan and Garcia-Molina, 1999]). Nous proposerons pour chaque structure un modèle analytique qui estime l'espace mémoire ainsi que le coût en temps de traitement. Ce modèle prend en compte le nombre de souscriptions et de termes présents dans la structure et les probabilités de présence de chaque terme dans l'index. Nous introduirons également la notion de satisfaction partielle des souscriptions par les items afin de notifier les souscriptions rarement satisfaites par le *broad match*.

4.2 Conception des index Pub/Sub

Comme nous l'avons vu dans la section 2.2.1, les structures d'indexation basées sur un compteur (*Count-based*) et celles basées sur un arbre (*Tree-based*) sont les principales structures d'indexation proposées dans la littérature pour compter (implicitement ou bien explicitement) le nombre de termes d'une souscription qui apparaissent dans un item. La majorité des travaux de recherche [Pereira et al., 2000a, Fabret et al., 2001, Aguilera et al., 1999] ne permettent pas le passage à l'échelle pour la conjonction des mots-clés dû à la taille importante du vocabulaire utilisé dans la syndication Web [Hmedeh et al., 2011b].

Nous proposons deux structures d'indexation basées sur les Listes Inverses [Zobel and Moffat, 2006] : la première utilise l'algorithme de comptage de mots-clés de la souscription dans l'item (*Count-based*) et la deuxième utilise les rangs des termes (*Ranked-key*). Enfin, nous détaillons une troisième structure qui est une variante de l'*Ordered Trie* (Treillis) [Knuth, 1973] avec des termes distincts dans les souscriptions indexées. Avant de présenter nos index, et pour mieux comprendre la sémantique de traitement d'un item dans les systèmes Pub/Sub, nous expliquons la sémantique du « *broad match* » entre les items et les souscriptions soumises.

4.2.1 Traitement des items

Dans les systèmes Pub/Sub pour la syndication Web, les utilisateurs soumettent des requêtes continues (souscriptions) sous la forme de mots-clés. Quand un nouvel item est publié, il est évalué avec toutes les souscriptions soumises au système pour trouver l'ensemble de souscriptions satisfaites qui seront notifiées par cet item. L'ensemble des souscriptions est noté $\mathcal{S}$ et leur nombre total par $|\mathcal{S}|$. Chaque souscription s dans $\mathcal{S}$ est composée d'un ensemble de termes distincts du vocabulaire de souscriptions $\mathcal{V}_S = \{t_1, t_2, ..., t_n\}$. La taille de s, notée $|s|$, est le nombre total de termes (distincts) dans s. $\mathcal{I} = [I_1, I_2, ..., I_m]$ est l'ensemble des items publiés dans le système. Un item $I \in \mathcal{I}$ est

aussi formé d'un ensemble de termes distincts[1] ($I \subseteq \mathcal{V}_I$, où $\mathcal{V}_I$ est le vocabulaire des items). Nous prenons la même hypothèse $\mathcal{V}_S \subseteq \mathcal{V}_I$ que [Yan and Garcia-Molina, 1994]. Toutefois, il est à noter que dans la réalité, la taille de $\mathcal{V}_S$ peut significativement dépasser $\mathcal{V}_I$. Dans ce contexte, une souscription s est satisfaite par un item I si et seulement si l'ensemble des termes de s est inclus dans l'ensemble des termes de I (*sémantique de « broad match »*, définition 1).

Définition 1 (Broad Match) *Un item $I \in \mathcal{I}$ satisfait une souscription $s \in \mathcal{S}$ si et seulement si $\forall t_j \in s \rightarrow t_j \in I$. Ayant un item I, le problème est de trouver l'ensemble de souscriptions $\mathcal{S}_M \subseteq \mathcal{S}$ qui sont satisfaites par I.*

Souscriptions	Termes
s_1	$t_1 \wedge t_2 \wedge t_4$
s_2	$t_1 \wedge t_3$
s_3	$t_1 \wedge t_2 \wedge t_5$
s_4	$t_2 \wedge t_4$
s_5	$t_1 \wedge t_3 \wedge t_6$

Table 4.1 – Exemple de souscriptions

Par la suite, nous utiliserons l'exemple du tableau 4.1 présentant un ensemble de souscriptions de mots-clés. Chaque souscription est une conjonction de termes précisés par l'utilisateur. Par exemple, pour s_1 l'utilisateur est intéressé par toutes informations contenant à la fois les termes t_1, t_2 et t_4. Alors que l'utilisateur, qui a souscrit s_4, s'intèresse à toute notification d'item contenant t_2 et t_4. L'ensemble des souscriptions satisfaites $\mathcal{S}_M$ par $I = \{t_2, t_4\}$ est égal à $\{s_4\}$, car t_2 et t_4 apparaissent dans I. s_1 n'est pas satisfaite par I comme il ne contient pas t_1. Une solution naïve pour traiter un item est de tester l'inclusion dans l'item de toutes les souscriptions soumises au système. Cette solution ne permet pas le passage à l'échelle pour des millions de souscriptions. L'indexation des souscriptions permettra d'éliminer rapidement les souscriptions qui ne sont pas satisfaites par l'item.

Les termes sont étiquetés par leur rang et sont classés suivant le nombre d'occurrences décroissant. Par exemple, t_1, le terme le plus fréquent (apparaît dans quatre souscriptions), possède le rang 1. Par suite t_2 qui apparaît dans trois est classé deuxième.

4.2.2 Listes inverses et Compteur

Les index Pub/Sub de type *Count-based Inverted List* (*CIL*) font correspondre pour chaque terme $t_j \in \mathcal{V}_S$ (vocabulaire des souscriptions) une liste inverse $Postings(t_j)$, c'est-à-dire la liste des souscriptions contenant ce terme. Ce vocabulaire est communément appelé *Dictionnaire*. Les souscriptions sont donc présentes dans plusieurs listes et une étape de recomposition de la souscription est nécessaire. C'est à cette fin qu'on définit une

1. Les termes dupliqués dans les items sont supprimés dans une phase de traitement antérieure.

structure supplémentaire : le *Compteur*, pour garder trace du nombre de termes qui doit être testé pendant le processus du traitement d'un item (*matching*). Pour le traitement d'un item, une copie de ce compteur est utilisée, une valeur par souscription correspond au nombre de termes restant pour qu'elle soit satisfaite.

La figure 4.1 illustre les souscriptions de l'exemple du tableau 4.1 indexées dans *CIL*. La liste inverse du t_2 $Postings(t_2) = \{s_1, s_3, s_4\}$. Le compteur est initialisé avec la taille de chaque souscription :

$$Compteur = \{s_1 : 3, s_2 : 2, s_3 : 3, s_4 : 2, s_5 : 3\}$$

Considérons l'item $I = \{t_2, t_4\}$, les valeurs du $Compteur$ de souscriptions de $Postings(t_2)$ et $Postings(t_4)$ sont décrémentées, et nous obtenons :

$$Compteur = \{\mathbf{s_1 : 1}, s_2 : 2, s_3 : 2, \mathbf{s_4 : 0}, s_5 : 3\}$$

s_4 sera alors notifiée puisque la valeur de son compteur est passée à 0.

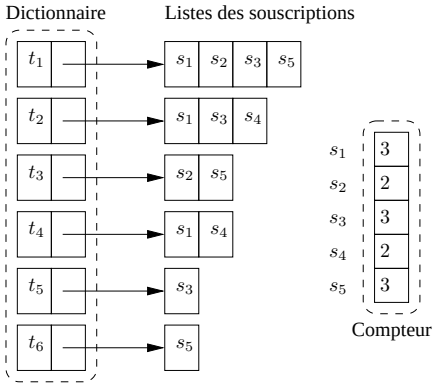

Figure 4.1 – Count-based Inverted List

CIL - Construction Pour chaque nouvelle souscription s, son identifiant sera ajouté dans toutes les listes inverses de ses termes. En plus, une entrée dans le *Compteur* lui sera associée correspondant à son nombre de termes distincts $|s|$.

CIL - Traitement (Matching) Le traitement d'un item I dans *CIL* est montré par l'algorithme 1. Pour déterminer l'ensemble de souscriptions satisfaites par I, une copie du *Compteur* est effectuée *Compteur_ copie*. Pour chaque terme t_j de I, sa liste inverse $Postings(t_j)$ est parcourue. Ensuite, pour chaque souscription s de $Postings(t_j)$, son compteur dans *Compteur_ copie* est décrémenté. Quand une valeur de *Compteur* passe à 0 la souscription correspondante est notifiée.

Algorithme 1: $CIL\_TRAITEMENT(I)$

Entrées : Un item I.

1: $Compteur\_copie \leftarrow$ copie du $Compteur$
2: **pour tout** terme $t_j \in I$ **faire**
3: $\quad Post \leftarrow Postings(t_j)$
4: $\quad$ **pour tout** $s \in Post$ **faire**
5: $\quad\quad Compteur\_copie[s] \leftarrow Compteur\_copie[s] - 1$
6: $\quad\quad$ **si** $Compteur\_copie[s] = 0$ **alors**
7: $\quad\quad\quad S_M \leftarrow S_M \cup \{s\}$
8: $\quad\quad$ **finsi**
9: $\quad$ **fin pour**
10: **fin pour**

4.2.3 Ranked-key Inverted List

L'index *Ranked-key Inverted List* (*RIL*) ajoute une souscription uniquement dans la liste inverse du terme le moins fréquent parmi tous ses termes. Ce terme est appelé *clé* (*key*) de la souscription. À part l'identificateur de la souscription, l'ensemble des termes restant de la souscription est ajouté à la liste inverse. Pour chaque terme d'un item publié, l'inclusion des souscriptions indexées dans sa liste inverse sera vérifée. Naturellement, les listes inverses des termes très fréquents sont réduites alors que celles des termes de fréquence moyenne deviennent plus larges.

La figure 4.2 présente l'index *RIL* de notre exemple où le rang des termes est donné par leur indice (t_1 est le terme le plus fréquent de $\mathcal{V}_S$). Nous remarquons l'absence des termes les plus fréquents du *Dictionnaire* de l'index. Par exemple, t_1 et t_2 n'apparaissent dans le dictionnaire car ils apparaissent toujours dans une souscription de $\mathcal{S}$ avec un terme moins fréquent. $Postings(t_4)$ contient deux souscriptions (s_1 et s_4) pour lesquelles le reste de leurs termes est stocké dans la liste. Pour $I = \{t_2, t_4\}$, son traitement commence par le test de l'inclusion des souscriptions de $Postings(t_4)$; pour $s_1 : I$ ne contient pas t_1 donc s_1 n'est pas satisfaite. Puis pour s_4, t_2 apparaît dans $I' = I - t_4 = \{t_2\}$ et il n'y a pas d'autres termes dans s_4 donc elle sera notifiée.

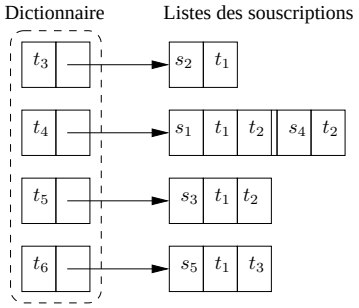

Figure 4.2 – Ranked-key Inverted List

RIL - Construction Lorsqu'une nouvelle souscription s est ajoutée au système, la liste inverse de son terme le moins fréquent est cherchée. Cela nécessite le tri des termes de s par leur rang. Puis une entrée correspondant à son identificateur est ajoutée dans la liste inverse, elle sera suivie par la liste des termes restant de s.

RIL - Traitement (Matching) Pour le traitement d'un item I, il faut d'abord trier ses termes par leur rang. Puis, la liste inverse du terme le moins fréquent $Postings(t_j)$ est traitée (Voir algorithme 2). Pour chaque souscription de $Postings(t_j)$, l'existence de chacun de ses termes dans I sera ensuite vérifiée.

Algorithme 2: $RIL\_TRAITEMENT(I)$

Entrées : Un item I.

```
1: termes_TR ← Trier(I)
2: tantque termes_TR ≠ {} faire
3:    t_j ← moins_fréquent(termes_TR)
4:    termes_TR ← termes_TR − t_j
5:    Post ← Postings(t_j)
6:    pour tout s ∈ Post faire
7:       si s.reste ⊆ termes_TR alors
8:          S_M ← S_M ∪ {s}
9:       finsi
10:   fin pour
11: fin tantque
```

4.2.4 Regular Ordered Trie

Une alternative pour les Listes Inverses repose sur un index de type *Ordered Trie*, capable de profiter des préfixes communs entre les souscriptions pour construire un espace de recherche hiérarchique (*c.-à-d.*, factorisation) (voir définition 2).

Définition 2 (Couverture des souscriptions) *Une souscription s_i couvre totalement une souscription s_j, noté par $s_i \succeq s_j$, si et seulement si $\forall t_j \in s_j \rightarrow t_j \in s_i$.*

Le lemme suivant détermine si deux souscriptions peuvent être factorisées :

Lemme 1 *Soit la fonction $\mathcal{ORDRE} : \mathcal{V} \mapsto \mathbb{I}$ qui présente un ordre total entre les termes du vocabulaire et associe à chaque terme t_j son rang dans $\mathcal{V}$. Deux souscriptions s_i et s_j partagent un préfixe si et seulement si $\forall t_k \in (s_i \cup s_j) \backslash (s_i \cap s_j), \forall t_l \in (s_i \cap s_j), \Rightarrow \mathcal{ORDRE}(t_l) < \mathcal{ORDRE}(t_k)$*

Dans le *Trie*, deux souscriptions partagent un préfixe si et seulement si elles ont les mêmes termes les plus fréquents. Nous détaillerons par suite l'impact de l'ordre sur la factorisation entre les souscriptions.

Un nœud du *Trie* représente un terme. Une souscription sera indexée dans le nœud n si et seulement si ses termes sont trouvés dans les nœuds du chemin de la racine jusqu'à n. Deux souscriptions qui partagent un préfixe de k nœuds sont fusionnées dans un chemin unique de longueur k (*c.-à-d.*, préfixe commun), suivi par deux chemins distincts représentant les termes restant des souscriptions. Nous pouvons comparer cette méthode aux *Tries* utilisés pour indexer les phrases d'un vocabulaire donné [Knuth, 1973], mais avec deux caractéristiques particulières pour l'*Ordered Trie* : (i) pas de répétition de termes dans les phrases (*c.-à-d.*, une souscription est un ensemble de termes) ; (ii) un ordre total est imposé aux termes. Cette structure nommée *Regular Ordered Trie* (ROT) a été utilisée dans différents domaines de recherche d'information [Yan and Garcia-Molina, 1994] et pour la fouille de données [Bodon, 2005, Malik and Kender, 2007], mais elle n'a jamais été utilisée pour les systèmes Pub/Sub.

La figure 4.3 illustre le ROT de l'exemple du tableau 4.1 où le rang du terme est donné par son indice (t_1 a le rang le plus élevé dans $\mathcal{V}_S$). Grâce à la factorisation, le ROT contient un seul nœud t_1 pour toutes les souscriptions qui le partage. Considérons l'item $I = \{t_2, t_4\}$ (dont les termes sont déjà triés), le terme t_2 est d'abord cherché parmi les fils de la racine. Comme il existe, le traitement de I continue en cherchant t_4 parmi les fils de t_2. Ce nœud existe, les souscriptions qui y sont stockées sont alors notifiées. Finalement, le dernier terme t_4 de I est traité, mais comme il n'existe pas un nœud qui lui correspond parmi les fils de la racine le traitement est terminé.

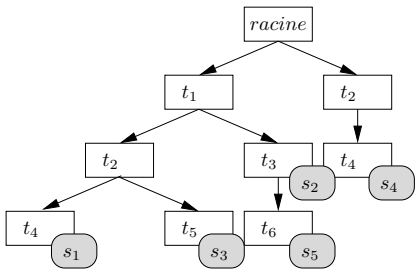

Figure 4.3 – Regular Ordered Trie

Afin de réduire le nombre de nœuds dans le *Trie*, ainsi que le nombre de nœuds visités durant le traitement d'un item, les sous-chemins pour lesquels chaque nœud a un unique fils peuvent être fusionnés en un seul nœud. Nous obtenons une variante du *Patricia Ordered Trie* (POT). Par exemple, les nœuds t_2 et t_4 de figure 4.3 sont fusionnés en un seul nœud $t_2.t_4$ du POT.

ROT - Construction Les termes d'une nouvelle souscription s sont tout d'abord triés par leur rang. Puis le chemin correspondant au premier terme de s est cherché depuis la racine. Ce traitement est répété pour chaque terme $t_j \in s$ (algorithme 3). Si un chemin particulier n'existe pas, un nouveau nœud étiqueté par le terme est ajouté au *Trie*. La souscription est stockée dans le nœud où la recherche de tous ses termes se termine.

Algorithme 3: $TRIE\_INDEXATION(s)$

Entrées : Une souscription s.

```
1: termes_TR ← Trier(s)
2: noeud_Courant ← racine
3: pour tout terme t_j ∈ termes_TR faire
4:    noeud_fils ← trouver le fils de noeud_Courant correspondant à t_j
5:    si noeud_fils = null alors
6:       noeud_fils ← nouveau nœud t_j
7:       Ajouter noeud_fils comme fils de noeud_Courant
8:    finsi
9:    noeud_Courant ← noeud_fils
10: fin pour
11: Ajouter s dans la liste du noeud_Courant
```

ROT - Traitement (Matching) Le processus de traitement d'un item I est donné par l'algorithme 4. À l'arrivée de I, ses termes sont triés et traités par ordre croissant. Puis tous les chemins qui correspondent aux termes les moins fréquents du nœud courant sont parcourus (lignes 5-10). Pour chaque nœud visité, les souscriptions qui y sont stockées sont alors notifiées (lignes 2-4).

Algorithme 4: $TRIE\_TRAITEMENT(N, I)$

Entrées : N : nœud courant du *Trie*, un item I

```
1: termes_TR ← Trier(I)
2: si N contient des souscriptions alors
3:    S_M ← S_M ∪ {s|s ∈ N}
4: finsi
5: pour tout terme t_j ∈ termes_TR faire
6:    noeud_fils ← trouver le fils correspondant à t_j
7:    si noeud_fils ≠ NULL alors
8:       TRIE_TRAITEMENT(noeud_fils, termes_TR − {t_j})
9:    finsi
10: fin pour
```

Trie complet

Deux paramètres influent sur la morphologie du *Trie* : (1) la taille des souscriptions et (2) la taille du vocabulaire. La taille des souscriptions impacte la profondeur du *Trie* : de larges souscriptions rendent le *Trie* plus profond. La taille du vocabulaire impacte de son côté la largeur du *Trie*. Ayant une taille maximale des souscriptions $|s|_{max}$ et un vocabulaire $\mathcal{V}_S$, un *Trie* complet (*Complete Ordered Trie*) est le *Trie* qui contient toutes les combinaisons possibles des souscriptions de taille $|s| \in [1, |s|_{max}]$ dont les termes appartiennent à $\mathcal{V}_S$. La figure 4.4 montre un COT où $|s|_{max} = 3$ et $\mathcal{V}_S = \{t_1, t_2, t_3, t_4\}$.

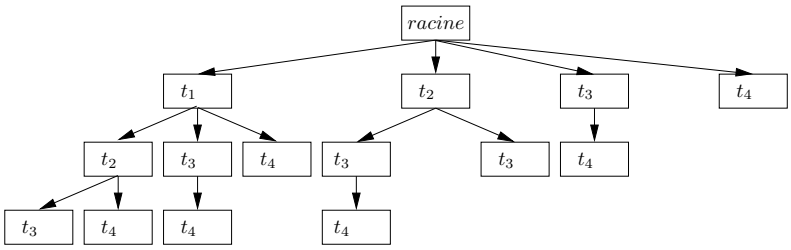

FIGURE 4.4 – Complete Ordered Trie

Ordre des termes et factorisation

Comme nous l'avons mentionné notre *Trie* est ordonné et la factorisation entre les souscriptions dépend de l'ordre appliqué aux termes du vocabulaire des souscriptions. Prenons les exemples de la figure 4.5(1) où les souscriptions $s_1 = \{t_1, t_2\}$ et $s_2 = \{t_1, t_3\}$ sont indexées, et de la figure 4.5(2) où les souscriptions $s_3 = \{t_1, t_3\}$ et $s_4 = \{t_2, t_3\}$ sont indexées. Dans la première l'ordre appliqué aux termes est l'ordre de leur nombre d'occurrences décroissant (t_1 est le plus fréquent) et dans la deuxième nous utilisons l'ordre inverse. Ces index sont le résultat de l'application du Lemme 1 qui définit la factorisation entre les souscriptions, donc dans le deuxième exemple pas de factorisation entre les souscriptions. Le gain de factorisation est maximal si l'ordre choisi pour les termes correspond aux nombres d'occurrences des termes ; comme les termes les plus fréquents auront les premiers rangs, ils apparaissent le moins dans l'index. Tandis que les termes qui ont des rangs très élevés, ayant un petit nombre d'occurrences ils n'occuperont pas tous les nœuds qui correspondent à leur rang dans le *Trie*. Par suite un *Trie* complet (COT) ne sera pas formé (voir section 4.5.3). Nous étudierons expérimentalement l'impact de l'ordre des rangs des termes sur la morphologie du *Trie* dans la section 4.5.3.

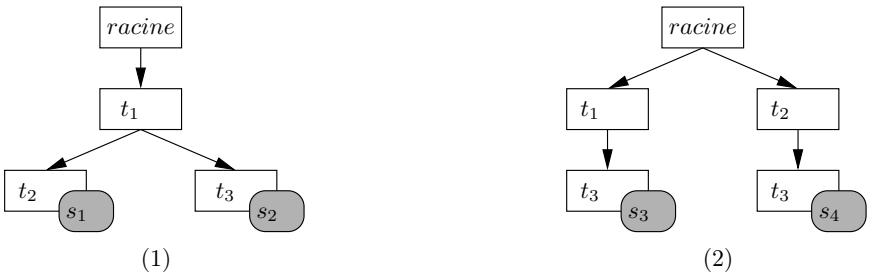

FIGURE 4.5 – Exemple de souscriptions pour différents rang des termes

Par la suite, nous utiliserons la notion de *nœud* dans l'index, même si elle n'a pas la même définition dans les trois structures. Dans *CIL*, un nœud représente l'identificateur de la souscription dans une liste inverse. Dans *RIL*, en plus de l'identificateur de la souscription s, le nœud peut contenir aussi d'autres identificateurs des termes restants de s. Dans *ROT*, un nœud dans la structure arborescente représente un terme du vocabulaire des

souscriptions.

4.3 Modèles analytiques

Les modèles analytiques que nous allons maintenant présenter pour nos index, nous permettent de prévoir l'espace mémoire et le temps de traitement de chacune de ces structures. Pour cela, nous mesurons le nombre de nœuds présents dans la structure ainsi que le nombre de nœuds visités durant le traitement d'un item.

$\|\mathcal{S}\|$	nombre total de souscriptions
$\|\mathcal{V}_I\|$, $\|\mathcal{V}_S\|$	taille des vocabulaires des items et des souscriptions
$\|s\|_{moy}$, $\|s\|_{max}$	resp. taille moyenne et maximale d'une souscription
$\|I\|$	taille d'un item
$P(t_j)$	distribution des fréquences des termes de $\mathcal{V}_I$
$\theta(k)$	probabilité qu'une souscription ait la taille k
σ_i	probabilité d'avoir un terme de rang $\leq i$
$w(c), w(v)$	taille d'une entrée dans le Compteur/Dictionnaire
$w(p), w(n)$	taille d'une entrée d'une liste inverse/nœud du Trie

Table 4.2 – Paramètres et notations

Le tableau 4.2 détaille l'ensemble des paramètres et des notations utilisés dans cette section. Le but de notre modèle est d'estimer le nombre de nœuds en fonction du nombre de souscriptions $|\mathcal{S}|$, du vocabulaire des items $|\mathcal{V}_I|$ et de celui des souscriptions $|\mathcal{V}_S|$. Les souscriptions elles-mêmes sont caractérisées par leur taille moyenne $|s|_{moy}$ et maximale $|s|_{max}$, et le vocabulaire par la distribution des fréquences $P(t_j)$ de chacun des termes qui le compose. Ensuite, il faut prendre en compte plusieurs probabilités conditionnelles ; $\theta(k)$: la probabilité qu'une souscription ait une taille k, et σ_i : la probabilité d'avoir un terme qui ait un rang $\leq i$, et par des tailles de nœuds dépendant de la structure considérée $w(c)$, $w(v)$ et $w(p)$ pour les listes inverses et $w(n)$ pour le *Trie*.

4.3.1 Temps d'indexation

Soit la taille moyenne d'une souscription définie par :

$$|s|_{moy} = \sum_{k=1}^{|s|_{max}} \theta(k) \times k,$$

où $\theta(k)$ est la probabilité qu'une souscription ait k comme taille avec $k \in [1, |s|_{max}]$. Nous pouvons aisément inférer que le temps nécessaire pour ajouter une souscription dans *CIL* est en $O(|s|_{moy})$. En effet, il faut ajouter $|s|_{moy}$ entrées dans les listes inverses et une dans le Compteur. Toutefois, l'index de type *RIL* nécessite un tri des termes de

la souscription pour déterminer le terme le moins fréquent. Ainsi, l'insertion est faite en $O(|s|_{moy} \times \log |s|_{moy})$.

Par ailleurs, le temps nécessaire pour ajouter une souscription dans un index de type ROT est en $O(|s|_{moy} \times \log |s|_{moy})$. Cela est dû à la nature arborescente et déterministe du ROT ainsi qu'à l'utilisation d'une implantation basée sur le hachage : le temps de tri des termes domine le temps de recherche ($O(1)$). Nous pouvons donc en déduire que le temps d'indexation dans tous les index est indépendant du nombre total des souscriptions $|\mathcal{S}|$ déjà présentes dans la structure.

4.3.2 Espace mémoire

L'espace mémoire pris par chacune des structures est estimé à l'aide d'un certain nombre de probabilités. Soit $P(t_j)$ la fréquence d'occurrence de $t_j \in \mathcal{V}_I$.

Les termes de chaque souscription sont choisis dans $|\mathcal{V}_I|$ d'une façon aléatoire en respectant leur distribution de fréquences, donc il est probable que les termes de $|\mathcal{V}_I|$ n'apparaissent pas dans toutes les souscriptions. Afin d'estimer le nombre de termes de $|\mathcal{V}_S|$ d'un index donné, nous avons défini $Pr(t_j \in s)$ et $Pr(t_j \in \mathcal{S})$. La probabilité qu'un terme t_j n'apparaisse pas dans une souscription de taille k est égale à la probabilité de tirer les k termes de la souscription différents de t_j. Connaissant $Pr(t_j \in s)$ nous pouvons définir $Pr(t_j \in \mathcal{S})$, donc pour qu'un terme appartienne à $|\mathcal{V}_S|$ il doit au moins apparaître dans une souscription de $\mathcal{S}$. La probabilité qu'un terme n'apparaisse pas dans $\mathcal{S}$, est le produit des probabilités qu'il ne soit choisi dans aucune souscription. Alors, la probabilité que t_j soit un des termes de la souscription s, notée $Pr(t_j \in s)$, et que t_j soit un terme dans au moins une souscription de $\mathcal{S}$ notée $Pr(t_j \in \mathcal{S})$ sont :

$$\begin{aligned}
Pr(t_j \in s) &= 1 - Pr(t_j \notin s) \\
&= 1 - \sum_{k=1}^{|s|_{max}} \theta(k) \times (1 - P(t_j))^k \qquad (4.1) \\
Pr(t_j \in \mathcal{S}) &= 1 - Pr(t_j \notin \mathcal{S}) \\
&= 1 - \prod_{i=1}^{|\mathcal{S}|} (1 - Pr(t_j \in s)) \\
&\overset{(4.1)}{=} 1 - (\sum_{k=1}^{|s|_{max}} \theta(k) \times (1 - P(t_j))^k)^{|\mathcal{S}|}
\end{aligned}$$

Ainsi la taille du vocabulaire des souscriptions $\mathcal{V}_S$ est égale à :

$$|\mathcal{V}_S| = \sum_{j=1}^{|\mathcal{V}_I|} Pr(t_j \in \mathcal{S})$$

Espace mémoire - CIL Le *CIL* comprend deux structures : le *Compteur* et les *listes inverses*, qui sont composées du *dictionnaire* et des *listes de souscriptions* (*Postings*). Donc l'espace mémoire occupé par l'index est égal à :

$$\begin{aligned} Taille(CIL) &= Taille(Dictionnaire) + Taille(Postings) \\ &\quad + Taille(Compteur) \end{aligned}$$

L'espace mémoire pris par le *Compteur* dépend du nombre de souscriptions :

$$Taille(Compteur) = |\mathcal{S}| \times w(c) \tag{4.2}$$

Chaque terme de $|\mathcal{V}_S|$ correspond à une entrée dans le *Dictionnaire* d'où la taille de ce dernier est égale à :

$$Taille(Dictionnaire) = |\mathcal{V}_S| \times w(v) \tag{4.3}$$

La taille des *listes de souscriptions* est estimée par :

$$Taille(Postings) = |\mathcal{S}| \times |s|_{moy} \times w(p) \tag{4.4}$$

L'espace mémoire pris par l'index est :

$$Taille(CIL) \overset{(4.2,4.3,4.4)}{=} |\mathcal{S}| \times w(c) + |\mathcal{V}_S| \times w(v) + |\mathcal{S}| \times |s|_{moy} \times w(p)$$

Nous aurons par ailleurs besoin de connaître pour chaque liste inverse le nombre de nœuds qu'elle contient. En fait, la taille de la liste de t_j est égale à son nombre d'occurrences. Sachant que le nombre total de termes tirés pour générer $|\mathcal{S}|$ souscriptions est $|\mathcal{S}| \times |s|_{moy}$ et que $\Pr(t_j \in S) / \sum_{k=1}^{|\mathcal{V}_S|} Pr(t_k \in S)$ est la fréquence normalisée de t_j dans l'ensemble de termes tirés, la taille de sa liste est donnée par l'équation 4.5.

$$Taille(Postings(t_j)) = \frac{Pr(t_j \in S)}{\sum_{i=1}^{|\mathcal{V}_S|} Pr(t_i \in S)} \times |\mathcal{S}| \times |s|_{moy} \tag{4.5}$$

Espace mémoire - RIL L'index *RIL* est composé des *listes des souscriptions* et du *Dictionnaire* dont les termes correspondent aux termes les moins fréquents de chaque souscription $s \in \mathcal{S}$.

$$Taille(RIL) = Taille(Dictionnaire) + Taille(Postings)$$

Donc la taille des *listes de souscriptions* est la même que celle dans *CIL* (Équation 4.4). Par contre, la taille du *Dictionnaire* est plus petite que celle de *CIL*. Elle dépend du nombre de termes les moins fréquents dans au moins une souscription.

Une souscription s est ajoutée dans la liste du terme t_j si et seulement si $t_j \in s$ et s'il n'existe pas un terme $t_i \in s$ avec $rang(t_i) < rang(t_j)$. Donc la probabilité que s soit dans la liste de t_j est :

$$Post(s, t_j) = \sum_{k=1}^{|s|_{max}} \theta(k) \times k \times P(t_j) \times (\sigma_{j-1})^{k-1}$$

avec σ_j, la probabilité d'avoir un terme de rang $\leq i$.

En effet, pour une souscription de taille k, il y a alors k possibilités pour choisir t_j, et les autres termes seront choisis parmi des termes plus fréquents. La probabilité d'avoir t_j dans le *Dictionnaire* est la probabilité d'avoir au moins une souscription dans sa liste, est égale à : $1-(1-Post(s,t_j))^{|\mathcal{S}|}$ qui est la probabilité que t_j soit le terme le moins fréquent pour au moins une souscription dans $\mathcal{S}$. Nous en déduisons que la taille du *Dictionnaire* est définie par :

$$Taille(Dictionnaire) \quad = \quad \sum_{j=1}^{|\mathcal{V}_I|} (1 - (1 - Post(s, t_j))^{|\mathcal{S}|}) \times w(v) \tag{4.6}$$

Et par conséquent, l'espace mémoire utilisé par *RIL* est :

$$Taille(RIL) \overset{(4.6,4.4)}{=} \sum_{j=1}^{|\mathcal{V}_I|} 1 - (1 - Post(s, t_j))^{|\mathcal{S}|} \times w(v) + |\mathcal{S}| \times |s|_{moy} \times w(p)$$

Sachant la probabilité que t_j soit le terme le moins fréquent dans une souscription s de taille k. Alors, la taille de sa liste de souscriptions est égale à la somme des probabilités telle que t_j soit le moins fréquent dans toutes les souscriptions de $|\mathcal{S}|$, multipliée par le nombre d'entrées k par souscription :

$$\begin{aligned} Taille(Postings(t_j)) &= |s|_{moy} \times \sum_{s \in \mathcal{S}} Post(s, t_j) \\ &= |\mathcal{S}| \times \sum_{k=1}^{|s|_{max}} k \times \theta(k) \times k \times P(t_j) \times (\sigma_{j-1})^{k-1} \end{aligned} \tag{4.7}$$

Espace mémoire - ROT Bien que des études analysant le *Trie* existent comme par exemple dans [Clément et al., 2001], dans la suite la première proposition à notre connaissance pour calculer le nombre de nœuds dans le *ROT*. Nos analyses prennent en considération la distribution des fréquences des termes et celle des tailles de souscriptions. Le temps de calcul de notre formule est exponentiel avec la taille des souscriptions et du vocabulaire. Pour cette raison, elle n'est applicable que pour de petits vocabulaires ($|\mathcal{V}_I| < 100$) et de courtes souscriptions ($|s| < 12$). Cependant elle peut être utilisée pour

estimer le nombre de nœuds visités dans le cas de traitement d'un item puisque la taille de son vocabulaire est limitée au nombre de termes de l'item.

Soit $\mathcal{P}$ le chemin de la racine vers un nœud étiqueté t_i. Son étiquette Λ est définie comme :

i) $\Lambda = \lambda$ si le chemin est vide.

ii) si $\mathcal{P}$ étiqueté Λ est le chemin de la racine jusqu'au nœud étiqueté t_i, alors $\Lambda \bullet j$ est l'étiquette du nœud contenant t_j dont le père est la fin du ($\mathcal{P}$) étiqueté t_i.

Par la suite, pour des raisons de simplicité, nous exprimons le préfixe de j par Λ. Soit s une souscription, il y a un chemin dans le *Trie* étiqueté par ses termes classés par l'ordre de leur rang. Il est à noter qu'il y a $C_k^{k-|\mathcal{P}|}$ possibilités de souscriptions qui partagent un préfixe Λ.

Soient s une souscription et $Q(\Lambda, j)$ la probabilité que le nœud à l'adresse $\Lambda \bullet j$ appartienne à s.

Lemme 2

$$Q(\Lambda, j) = \sum_{k=|\mathcal{P}|+1}^{|s|_{max}} \theta(k) \times Q(\Lambda, j, k)$$

où $Q(\Lambda, j, k)$ est la probabilité que le nœud à l'adresse $\Lambda \bullet j$ appartienne à une souscription de taille k et elle est égale à :

$$C_k^{k-|\mathcal{P}|} \times \prod_{m \in \mathcal{P}} P(t_m) \times P(t_j) \times (1 - \sigma_j)^{(k-|\mathcal{P}|-1)}$$

Preuve: La probabilité que t_j apparaisse dans s à l'adresse $\Lambda \bullet j$ est :

$$\prod_{m \in \mathcal{P}} P(t_m) \times P(t_j) \times (1 - \sigma_j)^{(k-|\mathcal{P}|-1)}$$

car les $k - |\mathcal{P}| - 1$ termes restant de la souscription doivent être parmi des termes moins fréquents que t_j (dans $\mathcal{V}_I - \{t_1, \ldots, t_j\}$) (rappel que σ_j est égale à la probabilité d'avoir un terme moins fréquent que t_j), et il y a $C_k^{k-|\mathcal{P}|}$ combinaisons possibles d'où nous obtenons l'équation précédente. □

Pour qu'un nœud n apparaisse dans le *Trie*, il faut que le terme n apparaisse dans au moins une souscription. Soit $P(\Lambda, j)$ la probabilité que le nœud n apparaisse à l'adresse $\Lambda \bullet j$. Alors, un nœud n est occupé avec une probabilité $P(\Lambda, j)$ égale à :

$$P(\Lambda, j) = 1 - (1 - Q(\Lambda, j))^{|\mathcal{S}|} \tag{4.8}$$

Pour finir, soit $E(\Lambda, j)$ le nombre estimé de nœuds dans le *Trie* de racine $\Lambda \bullet j$ (Λ est le préfixe de t_j).

Théorème 1 $E(\Lambda, j) = \sum_{m=i+1}^{|\mathcal{V}_I|-|s|_{max}+|\mathcal{P}|} P(\Lambda \bullet j, m) \times [1 + E(\Lambda \bullet j, m)]$

avec $E(\Lambda, j) = 0$ *si* $|\mathcal{P}| > s$ *où* $j \geq |\mathcal{V}_I|$

Le Théorème 1 permet d'estimer le nombre de nœuds du *Trie*, en comptant la probabilité d'existence de chacun des nœuds. Celle-ci est égale à la probabilité d'apparition du terme qui lui correspond dans une souscription. Il se base sur des appels récursifs du nœud père vers ses fils pour compter le nombre de nœuds dans leurs sous-arbres, tout en prenant en considération la probabilité de l'existence du nœud père. En effet, si un nœud à l'adresse $\Lambda \bullet i \bullet j$ est occupé, la taille du *Trie* de racine $\Lambda \bullet i \bullet j$ est égale à $E(\Lambda \bullet i, j)$.

Finalement, la taille du *Trie* est égale à :

$$Taille(Trie) = E(\lambda, 0) \times w(n)$$

.

4.3.3 Temps de traitement

Temps de traitement - CIL Le temps nécessaire pour traiter un item I dans *CIL* est égale à la somme du temps nécessaire pour copier le compteur, et celui de ses décrémentations. Le premier dépend du nombre de souscriptions indexées et le deuxième de la taille des listes de souscriptions des termes t_j de I (Équation 4.5). Le temps nécessaire pour décrémenter une valeur du compteur (resp. pour copier une entrée du compteur) est noté par τ_{decr} (resp. τ_{copie}). Le temps nécessaire pour traiter un item est :

$$\begin{aligned} TempsTraitement(CIL) &= Temps(Copie\_Compteur) + Temps(Postings) \\ &= |\mathcal{S}| \times \tau_{copie} + \sum_{j=1}^{|I|} Taille(Postings(t_j)) \times \tau_{decr} \\ &\overset{(4.5)}{=} |\mathcal{S}| \times \tau_{copie} + \left[\sum_{j=1}^{|I|} \frac{Pr(t_j \in \mathcal{S})}{\sum_{i=1}^{|\mathcal{V}_S|} Pr(t_i \in \mathcal{S})} \times |\mathcal{S}| \times |s|_{moy} \right] \times \tau_{decr} \end{aligned}$$

Temps de traitement - RIL Le temps nécessaire pour traiter un item I dépend du nombre de ses termes et de la taille de leur liste de souscriptions. D'abord, il faut trier les termes de I puis traverser les listes pour tester l'inclusion des souscriptions dans I. Le coût du traitement de I est estimé par :

$$
\begin{aligned}
TempsTraitement(RIL) &= Temps(Tri) + Temps(Postings) \\
&= |I| \times \log |I| + \sum_{j=1}^{|I|} Taille(Postings(t_j)) \times \tau_{test} \\
&\overset{(4.7)}{=} |I| \times \log |I| + \\
&\sum_{j=1}^{|I|} |\mathcal{S}| \times \sum_{k=1}^{|s|_{max}} k \times \theta(k) \times k \times P(t_j) \times (\sigma_{j-1})^{k-1} \times \tau_{test}
\end{aligned} \tag{4.9}
$$

où τ_{test} est le temps nécessaire pour tester l'inclusion d'un terme dans I.

Temps de traitement - ROT Afin d'estimer le nombre de nœuds visités dans ROT durant le traitement d'un item, nous allons utiliser le Théorème 1.

Considérons un ensemble de souscriptions S dont la taille respecte une distribution $Dist_k$ et utilise un vocabulaire $\mathcal{V}$, le ROT résultant de leur indexation est noté $T(S, Dist_k, \mathcal{V})$.

Définition 3 (Restriction d'un *Trie*) *La restriction $T'(S, Dist_k, \mathcal{V}') = \Delta_{V'}(T(S, Dist_k, \mathcal{V}))$ est un sous-trie du T sur le vocabulaire $\mathcal{V}' \subset \mathcal{V}$ avec la même profondeur que la taille maximale d'une souscription ($|s|_{max}$).*

En utilisant la définition de T' et le Théorème 1, nous pouvons calculer le nombre de nœuds visités pour traiter un item I.

Théorème 2 *Le nombre estimé de nœuds visités pour traiter un item I à travers un Trie T est égale au nombre de nœuds dans la restriction $\Delta_{\mathcal{V}(I)}(T)$ où $\mathcal{V}(I)$ est le vocabulaire de I.*

Ainsi, le nombre de nœuds visités pour traiter un item $I = \{t_1, \ldots, t_k\}$, qui sont classés par leur rang, en utilisant la formule $P(\Lambda, j)$ (Équation 4.8), est exprimé par :

$$
\begin{cases}
N = E(\lambda, 0) \quad avec \\
E(\lambda, j) = \sum\limits_{m=j+1}^{|I|-|s|_{max}+1} P(\lambda \bullet j, m) \times (1 + E(\Lambda \bullet j, m))
\end{cases}
$$

Cette formule a été validée avec un jeu réel de 5 000 items dont les occurrences des termes ($P(t_j)$) respectent une distribution *Zipf* et un million de souscriptions indexées de taille constante. Les résultats des tests faits avec plusieurs jeux de souscriptions de différentes tailles montrent une déviation maximale de $-1,27\%$ lorsque la taille des souscriptions est égale à 8 (voir tableau 4.3). Avec de larges souscriptions, la profondeur du *Trie* est plus élevée, d'où une approximation du calcul qui se trouve propagée à chaque niveau. Pour un nombre plus important de nœuds, la somme de l'approximation est plus significative.

$\|s\|_{max}$	réel	estimé (Théorème 1)	déviation (%)
2	426, 7	426, 89	+0, 043
3	538, 99	538, 86	−0, 024
4	576, 52	575, 77	−0, 13
5	594, 66	590, 03	−0, 77
6	600, 15	595, 64	−0, 75
7	603, 36	596, 06	−1, 2
8	599, 8	592, 17	−1, 27

Table 4.3 – Nombre de nœuds visités (réel et estimé)

Pour conclure, le temps total de traitement d'un item est la somme du temps nécessaire pour trier ses termes et des temps utilisés au niveau de chaque nœud visité (noté par τ_n), et est égal à :

$$\begin{aligned} TempsTraitement(ROT) &= Temps(Tri) + Temps(Noeuds) \\ &= |I| \times \log|I| + E(\lambda, 0) \times \tau_n \end{aligned} \tag{4.10}$$

4.4 Satisfaction partielle des souscriptions

Nous avons déjà vu dans le chapitre 3 que d'après la distribution des fréquences des termes du vocabulaire publié sur le Web 2.0, il y a un grand nombre des termes qui n'apparaissent qu'une seule fois (voir section 3.5.1). Selon la définition de *broad match* pour chercher les souscriptions qui sont satisfaites par un item publié (Voir Définition 1), il faut que tous les termes de la souscription apparaissent dans l'item pour notifier l'utilisateur. Alors si un seul terme de la souscription n'apparaît pas dans l'item, l'utilisateur ne sera pas notifié. Par cette condition, l'utilisateur peut manquer beaucoup de l'information qui pourrait l'intéresser. Pour cela, nous proposons un « *Partial match* » ou encore appelée « Satisfaction partielle » de la souscription pour notifier l'utilisateur.

Dans cette section nous développons notre proposition de satisfaction partielle des souscriptions en adaptant nos trois structures d'indexation déjà prśentées. La recherche des souscriptions qui sont partiellement satisfaites se base sur les poids des termes de la souscription qui apparaissent dans l'item. En fait, il faut associer à chaque terme de la souscription un poids d'importance et définir un seuil de satisfaction. Si la somme des poids des termes de la souscription qui apparaissent dans l'item dépasse le seuil de satisfaction l'utilisateur sera notifié. Avec cette définition de satisfaction des souscriptions, l'utilisateur recevra les items qui contiennent les termes les plus importants de ses souscriptions.

Nous présentons en premier la technique utilisée pour pondérer les termes de la souscriptions et par suite les approches utilisées dans nos index pour chercher les souscriptions partiellement satisfaites.

4.4.1 Pondération des termes

La satisfaction partielle d'une souscription par un item se base sur un taux de satisfaction (score). Pour calculer le taux de satisfaction d'une souscription par un item publié, selon ses termes qui apparaissent dans l'item, nous donnons à chaque terme de la souscription un poids qui présente son importance. Nous pondérons tous les termes de notre vocabulaire globale acquis dans le chapitre 3 noté par $\mathcal{V}_I$ en se basant sur les informations présentées dans les items, par conséquent $\mathcal{V}_S \subseteq \mathcal{V}_I$.

Plusieurs technique de pondération sont proposées dans la littérature comme TF-IDF qui se base sur la fréquence du terme (*Term Frequency* (TF)) et le nombre de documents qui le contiennent *Inverse Documents Frequency* (IDF) [Baeza-Yates and Ribeiro-Neto, 1999], *Term Discrimination Value* (TDV) [Salton et al., 1975] ou bien la précision du terme (*Term Precision*) [Bookstein and Swanson, 1974]. Cependant dans notre contexte les items sont de petites tailles avec un petit nombre d'occurrences des termes. Nous considérons qu'une mesure traditionnelle comme TF-IDF n'est pas adéquate pour mesurer les poids des termes dans les items. Pour cela nous avons choisi *TDV* comme fonction de pondération. Elle mesure l'importance du terme comme étant sa capacité de distinguer entre les items (*c.-à-d.*, l'influence du terme sur l'entropie globale de l'ensemble des items). Pour notre ensemble d'items (et souscriptions), ni les termes les très fréquents (qui apparaissent dans beaucoup d'items, et donc ne sont pas des termes sélectifs et ne permettent pas de distinguer les items les uns des autres), ni les termes rares (présents dans un petit nombre d'items, et qui, s'ils ne sont pas de fautes d'orthographe, ne permettront probablement pas la satisfaction des souscriptions) n'auront pas une valeur importante de TDV.

Plus précisement, la valeur TDV d'un terme t_k est la différence entre la densité de l'ensemble d'items et la densité de cet ensemble sans t_k. Considérons que $sim(I_1, I_2)$ est la fonction de similarité utlisée pour mesurer la similarité entre les items (par exemple Cosinus ou bien distance Euclidienne) nous calculons la densité de l'ensemble d'items comme étant la similarité item-item moyenne :

$$\Delta(\mathcal{I}) = \frac{1}{|\mathcal{I}| \times (|\mathcal{I}| - 1)} \sum_{i=1}^{|\mathcal{I}|} \sum_{j=1 \wedge j \neq i}^{|\mathcal{I}|} sim(I_i, I_j)$$

Finalement la valeur de TDV d'un terme t_k est égale à :

$$tdv(\mathcal{I}, t_k) = \Delta(\mathcal{I} - \{t_k\}) - \Delta(\mathcal{I})$$

Par suite et pour simplifier, nous utilisons $tdv(t_k)$ au lieu de $tdv(\mathcal{I}, t_k)$.

En se basant sur les valeurs de TDV des termes calculées sur l'ensemble de nos items pour le vocabulaire $\mathcal{V}_I$ (rappelons que $\mathcal{V}_S \subseteq \mathcal{V}_I$), nous calculons les poids des termes des souscriptions. Le poids d'un terme dans une souscription est sa valeur de TDV normalisée par la somme des valeurs de TDV des termes de la souscription.

Définition 4 (Poids d'un terme dans une souscription) *Soit* $s = \{t_1, t_2, \ldots, t_n\}$ *une souscription, le poids d'un terme* $t_k \in s$ *noté par* $\varrho(t_k, s)$ *est égal à :*

$$\varrho(t_k, s) = \frac{tdv(t_k)}{\sum_{t_i \in s} tdv(t_i)}$$

À noter que notre proposition pour la satisfaction partielle des souscriptions et l'adaptation de nos index sont indépendantes de la fonction de pondération utilisée. TDV peut être remplacé par une autre mesure des poids sans aucun impact sur la structure et le traitement des items.

4.4.2 Approches de traitement partiel

Dans cette section, nous présentons nos approches d'indexation des souscriptions pour une satisfaction partielle par les items publiés. Avant nous définissons formellement la satisfaction partielle des souscriptions.

Supposons l'existence d'un seuil de satisfaction κ, ayant un item I il faut chercher les souscriptions tel que leur score de satisfaction soit supérieur à κ. Le score de satisfaction d'une souscription s noté par $\mu(s, I)$ est défini comme la somme des poids des termes de s qui apparaissent dans I, et il est égal à :

$$\mu(s, I) = \sum_{t \in s \cap I} \varrho(t, s)$$

Par suite, le traitement d'un item consiste à chercher les souscriptions qui satisfont la définition de satisfaction partielle donnée par la Définition 5.

Définition 5 (Satisfaction partielle) *Un item* $I \in \mathcal{I}$ *satisfait partiellement une souscription* $s \in \mathcal{S}$ *si et seulement si* $\mu(s, I) = \sum_{t \in s \cap I} \varrho(t, s) \geq \kappa$

Décomposition des souscriptions

La première approche consiste à indexer pour chaque souscription tous les sous-ensembles de ses termes dont le score de satisfaction est supérieure à κ. Par conséquence, dans le pire des cas, pour une souscription s nous avons $2^{|s|} - 1$ sous-souscriptions dont le score de satisfaction dépasse κ (*c.-à-d.*, l'ensemble de toutes les combinaisons possibles des termes de s). Afin de réduire le nombre de sous-souscriptions à indexer nous utilisons la propriété d'inclusion des souscriptions suivante :

Propriété 1 (Propriété d'inclusion des souscriptions) *Soient* s' *et* s'' *deux sous-souscriptions de* s *qui satisfont le seuil de satisfaction de* s*. Si* $s'' \subset s'$ *et* $\mu(s'', I) \geq \kappa$*, alors* $\mu(s', I) \geq \kappa$*. En d'autres termes, toutes satisfactions de* s' *qui permettent de notifier* s *sont déjà couvertes par* s''*.*

Par conséquent, nous indexons pour chaque souscription s seulement les sous-souscriptions qui satisfont le seuil de satisfaction de s et qui ne contiennent aucune sous-souscriptions satisfaisant le seuil de satisfaction.

Cette solution est nécessaire pour le *RIL* et le *POT*, ces index ne permettent pas de stocker des informations concernant le seuil de satisfaction de la souscription. En fait, dans *RIL* l'absence du terme clé de la souscription dans les termes de l'item, empêche sa notification même s'il existe un sous-ensemble de ses termes satisfaisant son seuil de satisfaction. Dans *POT*, les chemins communs entre les souscriptions dus à la factorisation ne permettent pas d'attribuer des poids aux termes pour chaque souscription.

L'approche de décomposition des souscriptions permet de créer tous les sous-ensembles de la souscription satisfaisant le seuil de satisfaction et les indexer. De plus, cette approche permet la classification des items notifiés par ordre de pertinence si nous stockons pour chaque sous-souscription son seuil de satisfaction.

Count-based Inverted List pour une satisfaction partielle

Nous proposons une extension du *CIL* pour prendre en considération la satisfaction partielle des souscriptions appelée CIL_p. La structure de CIL_p identique à celle de *CIL*, mais nous stockons en plus pour chaque terme son poids dans la souscription. D'où le besoin d'une entrée suplémentaire dans les listes de souscriptions de chaque terme. Une entrée dans la liste des souscriptions correspond à un couple $[s_i, \varrho(t_j, s_i)]$ où s_i est l'identificateur de la souscription qui contient le terme t_j et $\varrho(t_j, s_i)$ est son poids dans s_i.

Pour traiter un item dans CIL_p, il faut compter le score de satisfaction de la souscription qui est égal à la somme des poids de ses termes qui apparaissent dans l'item. Le Compteur du nombre de termes de *CIL* est remplacé par un `Compteur cumulatif` des poids des termes de la souscription qui apparaissent dans l'item. Dans ce compteur, nous attribuons une entrée par souscription, initialisée à 0, et la souscription sera notifiée si le score de satisfaction cumulé par les poids de ses termes atteint son seuil de satisfaction κ. À noter que nous aurons plus besoin de faire la copie du compteur pour traiter un item, il suffit de créer un compteur par item à traiter. Pour traiter un item, il faut parcourir les listes des souscriptions des termes $t_j \in I$ et pour chaque souscription s dans $Postings(t_j)$, incrémenter sa valeur dans le compteur par le poids de t_j dans s $\varrho(t_j, s)$. Quand la valeur du compteur dépasse le seuil de satisfaction κ, la souscription est considérée comme satisfaite.

$\mathbf{CIL}_{pu}$: variante de $\mathbf{CIL}_p$

Dans le but de permettre à l'utilisateur de choisir le seuil de satisfaction pour ses souscriptions, nous proposons CIL_{pu} une variante de CIL_p. Dans cette variante, comme dans *CIL*, le Compteur doit stocker l'information de chaque souscription s_i, *c.-à-d.*, son seuil de

satisfaction κ_i. Donc pour traiter un item dans CIL_{pu}, il faut faire une copie du compteur, et pour chaque terme $t_j \in I$, pour chaque souscription s dans $Postings(t_j)$, décrémenter sa valeur dans la copie du Compteur par $\varrho(t_j, s)$ le poids de t_j dans s . Quand la valeur dans le Compteur atteint 0, la souscription est considérée comme satisfaite.

Nous avons présenté les structures d'index Pub/Sub et leur modèle analytique pour estimer leur taille et le temps nécessaire pour traiter un item. Nous avons également proposé des approches pour nos index qui permettent la recherche des souscriptions qui sont partiellement satisfaites par les items publiés. Nous allons maintenant mesurer l'impact de plusieurs paramètres (taille des souscriptions, ordre des termes, distribution des fréquences des termes) sur leur morphologie (nombre de nœuds). Nous présenterons aussi un ensemble d'expériences dont le but est d'évaluer les performances des index en espace mémoire et temps de traitement tout en faisant varier la taille du vocabulaire des souscriptions ainsi que le nombre de souscriptions indexées.

4.5 Expériences : Evaluation des index

Cette section présente l'évaluation des index déjà présentés dans la section 4.2. Pour cela, différentes expériences seront effectuées pour mettre en valeur chacune de leurs caractéristiques. Tout d'abord, nous étudierons la morphologie de nos structures en nombre de nœuds et l'impact de différents paramètres, ce qui nous permettra de valider le modèle analytique proposé. L'impact de la taille des souscriptions sur le nombre de nœuds dans le *Trie* est présenté. Ainsi que l'impact de la distribution du nombre d'occurrences des termes et l'ordre des termes utilisé sur le nombre de nœuds des index et le nombre de nœuds visités pour le traitement d'un item dans chaque index. Par la suite, nous étudierons dans un cas réel l'espace mémoire utilisé par les index ainsi que leurs performances en temps de traitement en faisant varier différents paramètres. Avant de présenter les résultats de nos expériences, nous détaillerons l'implantation des index et les jeux de données utilisés.

4.5.1 Implantation

Les index sont implantés en utilisant la plateforme standard de Java (*Java version* « 1.6.0_20 »). Toutes les expériences ont été exécutées avec un processeur quadri-cœur 3,60 GHz et un espace mémoire de 16 Go pour la JVM. Nous décrivons l'implantation de chaque index et la motivation derrière le choix des structures de données utilisées et de leurs différents paramètres.

Les index basés sur les listes inverses (*CIL*, *RIL*) sont constitués de deux composants : le *Dictionnaire* qui présente les termes du vocabulaire des souscriptions et les listes des identificateurs des souscriptions indexées. Le *Dictionnaire* est implanté en utilisant une table de hachage statique qui hérite de la table de hachage *HashMap* de Java (voir

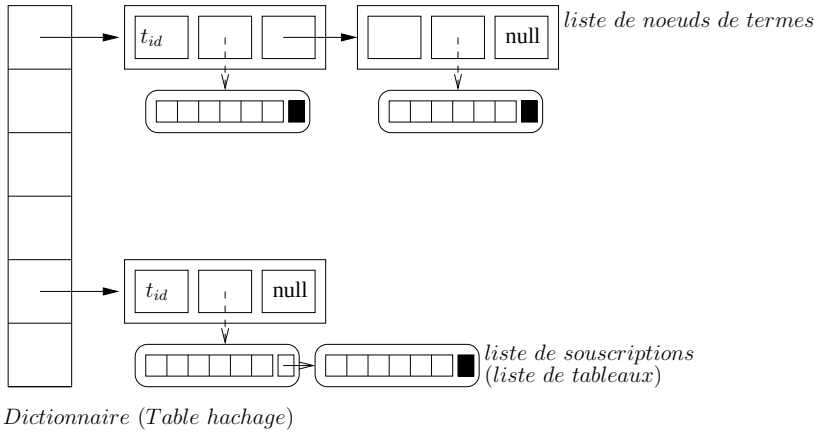

Figure 4.6 – Implantation des index basés sur les listes inverses (*CIL*, *RIL*)

figure 4.6). Les listes des identificateurs des souscriptions sont implantées comme des listes chaînées de tableaux. Les identificateurs des souscriptions sont codés comme des entiers sur 4 octets (*int*). Grâce aux collisions dans ces index, une entrée dans le dictionnaire peut correspondre à plusieurs termes. Pour cette raison, pour chaque entrée est associée une liste chaînée des nœuds des termes. Chaque nœud présente un terme et il est composé de : $(terme_{id}, \uparrow subs_{liste}, \uparrow suivant)$, où $\uparrow subs_{liste}$ est un pointeur vers la liste des identificateurs des souscriptions implantée comme une liste des tableaux et $\uparrow suivant$ est un pointeur vers le prochain nœud dans la liste. Finalement, le *Compteur* dans *CIL* est un tableau de *bytes* (octets) en nombre de souscriptions indexées $|\mathcal{S}|$ (la taille d'une souscription est beaucoup plus petite que 2^8 termes). Dans *RIL*, les identificateurs des termes de la souscription sont stockés séquentiellement après l'identificateur de la souscription.

Des expériences réalisées [Hmedeh, 2010] afin de déterminer les meilleures valeurs pour plusieurs paramètres comme la taille de la table de hachage statique (*Dictionnaire*) ainsi que la taille des tableaux des listes des identificateurs de souscriptions. Les résultats obtenus montrent que fixer la taille de la table de hachage pour le *Dictionnaire* à la moitié de la taille du vocabulaire des souscriptions $\mathcal{V}_S$ est un bon compromis pour minimiser le nombre de collisions dans la table sans augmenter le nombre des entrées non utilisées.

Des expériences, faites pour différentes tailles de jeux de souscriptions, montrent que nous obtenons le meilleur compromis entre le nombre d'objets créés pour les listes des identificateurs des souscriptions et le nombre des entrées vides dans les tableaux de ces objets quand la taille des tableaux est fixée à 25 éléments.

Plusieurs structures de données (par exemple : listes inverses, tableaux ou arbres) ont été proposées pour une implantation efficace du *Trie*. Nous avons choisi une implantation arborescente basée sur le hachage [Bagwell, 2000] pour nos *Tries* (ROT, POT) (Voir figure 4.7).

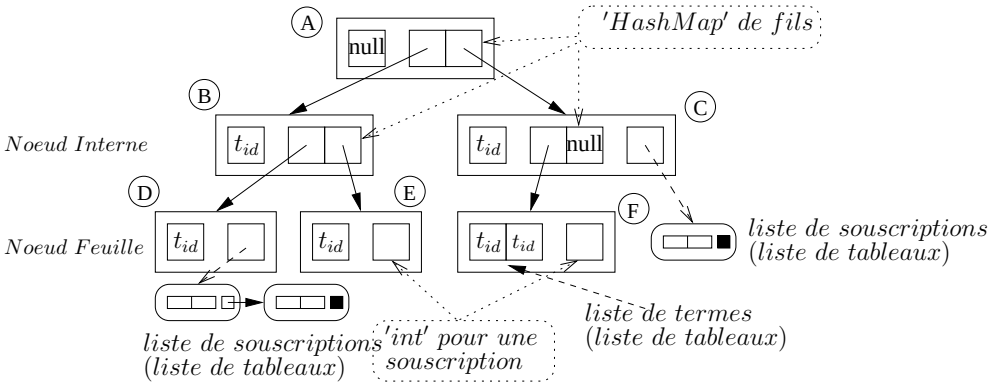

Figure 4.7 – Implantation des index basés sur les arbres (ROT, POT)

Chaque nœud interne est composé de : (i) un entier de 4 octets pour coder le rang du terme, (ii) une liste pour stocker les identificateurs des souscriptions correspondantes (la même structure de la liste utilisée pour CIL et RIL), et (iii) une table de hachage Java (*HashMap*) pour sauvegarder les nœuds correspondant à ses fils. Un nœud feuille contient seulement un entier pour le rang du terme et une liste pour les identificateurs des souscriptions. Cette optimisation effectuée entre la structure des nœuds internes et les feuilles est motivée par le fait que nous avons observé qu'il y a un nombre important de nœuds-feuilles. En plus, pour des raisons d'optimisation d'espace mémoire, nous avons distingué plusieurs types de nœuds en fonction de nombre de fils et de l'existence des souscriptions affectées. Nous distinguons alors les nœuds de ceux sans souscriptions, ainsi que ceux sans fils, avec un seul ou avec plusieurs fils. Pour chaque type de nœud une implantation est proposée.

Par exemple, le nœud B (figure 4.7) est un nœud interne avec une table de hachage pour indexer ses fils, mais il ne contient pas de liste de souscriptions, contrairement à C. À noter que les nœuds-feuilles comme D ou E ne contiennent pas de table de hachage comme ils n'ont pas de fils. Alors que des nœuds peuvent contenir une seule souscription, nous avons réduit l'espace utilisé en utilisant un seul entier pour l'identificateur de la souscription (nœud E) à la place de la liste des tableaux (nœud D).

Pour finir, dans POT, les chemins des nœuds un-aires (qui ne contiennent pas de listes des souscriptions et n'ont qu'un seul fils) sont fusionnés dans un seul nœud *compact*, étiqueté par l'ensemble de termes des nœuds fusionnés. Nous utilisons un tableau d'entiers de 4 octets pour stocker les rangs de ces termes (nœud F). Malgré le gain de factorisation en nombre de nœuds dans le *Trie* (voir section 4.5.3), l'espace mémoire utilisé par un nœud de POT est clairement beaucoup plus important que celui utilisé par un nœud dans CIL et RIL : un nœud dans CIL/RIL occupe seulement 4 octets, alors qu'un nœud du POT occupe en moyenne 128 octets (un objet en Java avec une table de hachage, pointeurs, listes des tableaux, etc.).

4.5.2 Description des jeux de données

Les expériences sont réalisées avec notre jeu de données réelles de flux RSS acquis pendant une période de huit mois de mars à octobre 2010. Ce jeu de données a été étudié et caractérisé (voir chapitre 3). Un nombre total de 10,7 millions d'items sont collectés provenant de 8 155 flux. Les items acquis sont stockés dans une base de données MySQL, où pour chaque flux nous avons gardé une seule occurrence de chaque item [Hmedeh et al., 2011b].

Du contenu textuel des items, un vocabulaire de 1,5 millions de termes distincts a été extrait et utilisé pour la génération des souscriptions. Nous avons utilisé la méthode Alias [Walker, 1977] pour la génération des souscriptions qui permet la génération en précisant la distribution des occurrences des termes du vocabulaire $Dist$. Trois distributions sont choisies pour les souscriptions de nos expériences : *réelle* (la distribution des occurrences des termes dans les souscriptions respecte la même distribution que celle des termes dans le jeu réel d'items), *uniforme* (les termes du vocabulaire sont utilisés avec la même probabilité de sélection), et *inversée* (la distribution des occurrences des termes dans les souscriptions respecte la distribution inverse de celle des termes dans les items). La génération des souscriptions est caractérisée par trois paramètres : (a) la taille du vocabulaire $\mathcal{V}_I$ utilisé et la distribution des occurrences des termes à respecter dans les souscriptions, (b) le nombre de souscriptions à générer $|\mathcal{S}|$, et (c) la taille des souscriptions k qui peut être constante pour toutes les souscriptions, ou bien respecter une distribution particulière. Dans nos expériences, quand la taille des souscriptions n'est pas précisée, nous utilisons la distribution des tailles des requêtes sur le Web [Vouzoukidou, 2010, Beitzel et al., 2004]. Elle est caractérisée par une taille maximale de 12 et une taille moyenne égale à $2,2$.

4.5.3 Morphologie des index

Dans cette section, nous étudierons l'impact de la taille des souscriptions et de la distribution des fréquences des termes sur la taille des index, en nombre de nœuds. Ces paramètres permettent de mettre en valeur le degré de factorisation entre les souscriptions du ROT (ou POT) par rapport au CIL (ou RIL). Ce gain de factorisation, apparaît dans la figure 4.8 qui représente le nombre de nœuds par rang de terme du ROT et celui du CIL. Pour cela, nous avons indexé 10 millions de souscriptions avec un vocabulaire $|\mathcal{V}_S| = 470\ 000$ dont la distribution des fréquences d'occurrences de ses termes suit la distribution réelle des termes des items [Hmedeh et al., 2011b]. Il apparaît que les tailles des listes des souscriptions dans CIL sont égales aux fréquences des termes. Nous observons que le nombre de nœuds du ROT a considérablement réduit par rapport à celui du CIL (dû à la factorisation). Il en va de même pour le *Trie* complet (*Complete Ordered Trie-COT*) qui contient toutes les combinaisons possibles des souscriptions pour un vocabulaire donné. Nous pouvons également remarquer que le gain en nombre de nœuds diminue avec le rang du terme.

Le tableau 4.4 montre le gain obtenu en nombre de nœuds pour plusieurs rang de terme

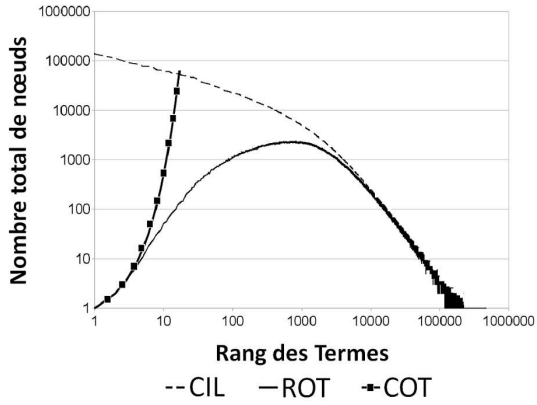

Figure 4.8 – Nombre de nœuds par rang de terme

Rang	# occurrences	# nœuds ROT	Gain (%)
1	138 090	1	99, 99
10	60 469	52	99, 91
1000	4 967	2 201	55, 69
10000	251	218	13, 15
470000	1	1	0

Table 4.4 – Gain par rang de terme

tr. Plus précisement, un gain est défini comme $G_{tr} = (N_{tr}(CIL) - N_{tr}(ROT))/N_{tr}(CIL)$, où $N_{tr}(CIL)$ et $N_{tr}(ROT)$ sont respectivement le nombre de nœuds dans CIL et ROT pour le terme de rang tr. Le nombre d'occurrences d'un terme est le nombre de nœuds qui lui correspondent dans CIL. Comme espéré, le gain diminue avec le rang du terme : de 1 pour le rang 1 (le terme le plus fréquent) à 0 pour le rang 470 000 (pas de factorisation). Il n'y a pas de factorisation quand le nombre de nœuds est le même dans ROT et CIL, c'est le cas pour les termes qui sont peu fréquents et apparaissent comme des feuilles dans ROT. Pour tous les termes ayant un rang supérieur à 18 789, le gain est égal à 0.

Taille des souscriptions

Pour étudier l'impact de la taille des souscriptions sur la taille du ROT, nous avons comparé le nombre de nœuds par rang de terme dans différents jeux de souscriptions de différentes tailles. Nous avons fixé le nombre total de termes dans les souscriptions à $1,5$ millions, d'où un nombre de souscriptions de chaque ensemble égal à $1,5M/k$ avec $k \in \{3, 6, 9, 12, 24, 36\}$. En faisant varier la taille des souscriptions, nous trouvons que cette dernière a un impact sur leur factorisation. Ainsi, le nombre de nœuds du ROT augmente avec la taille des souscriptions (voir figure 4.9), et par conséquent la factorisation diminue avec celle-ci. En effet, la probabilité que deux souscriptions partagent le même préfixe diminue avec leur taille surtout que le nombre de souscriptions indexées diminue quand

la taille des souscriptions augmente.

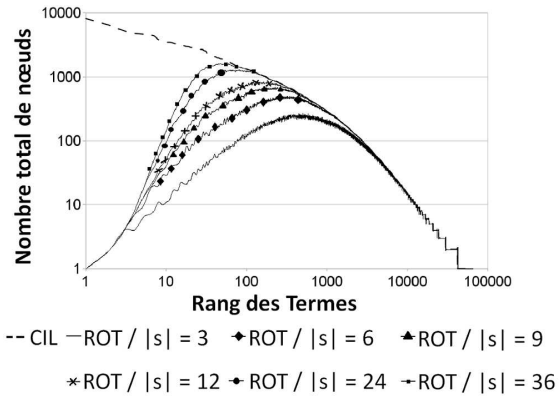

Figure 4.9 – Nombre de nœuds par rang de terme pour différentes tailles de souscriptions

Ordre des termes

Pour mesurer l'impact de l'ordre des termes sur la morphologie du *ROT* et du *POT*, nous avons fait des expériences avec différents ordres de termes : (a) *ordre des fréquences* (ordre décroissant des occurrences des termes dans les souscriptions), (b) *ordre inversé* (ordre croissant des occurrences des termes dans les souscriptions), (c) *ordre d'arrivée* (ordre d'apparition des termes dans les souscriptions),(d) *ordre aléatoire* (le rang des termes est choisi aléatoirement).

Les résultats montrent que l'ordre des termes n'a pas un impact significatif sur le nombre de nœuds dans les *Tries*. En effet, une augmentation du nombre de nœuds dans *ROT* peut être constatée en passant de l'ordre des fréquences des termes à son inverse. Avec un ordre inversé, la factorisation au début du *Trie* ne sera pas importante et plus de simples chemins apparaissent au fond du *ROT*. Dans *POT*, ces simples chemins seront fusionnés dans un seul nœud d'où pas d'augmentation remarquable du nombre de nœuds dans le *Trie*.

Concernant le nombre de nœuds visités du *ROT*, il est plus important quand l'ordre des termes est celui de leur fréquence que dans l'ordre inversé. En effet, l'ordre inversé permet d'arrêter plus tôt le parcours du *Trie*, puisque les termes les moins fréquents n'apparaissent pas beaucoup. Enfin, l'ordre des termes a un impact presque négligeable sur la taille et le nombre de nœuds visités du *POT*.

Distribution des fréquences

Nous allons maintenant nous intéresser à l'impact de la distribution des fréquences des termes dans les souscriptions : (a) *uniforme*, (b) *inversée* où le terme le plus fréquent

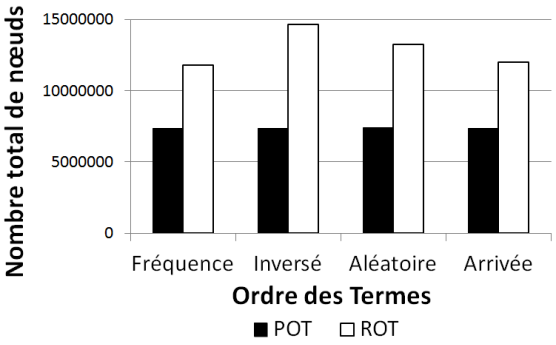

Figure 4.10 – Impact de l'ordre des termes sur le nombre de nœuds dans les index

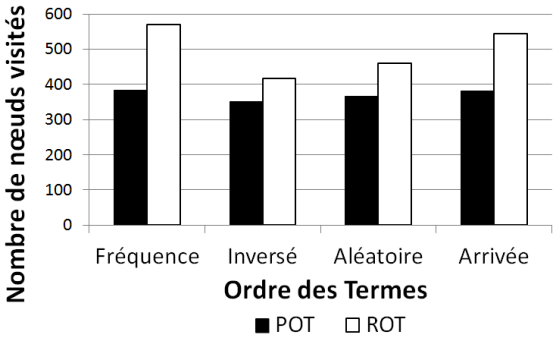

Figure 4.11 – Impact de l'ordre des termes sur le nombre de nœuds visités

dans les souscriptions correspond au terme le moins fréquent dans les items. À noter que les termes sont classés selon leur *fréquence* dans les items.

Dans le *CIL* et le *RIL*, le nombre de nœuds dépend uniquement du nombre de termes des souscriptions, et il est donc indépendant de la distributions des fréquences. Concernant le *ROT*, une distribution réelle ou inversée n'a pas d'impact sur la taille de l'index. En fait, il y a le même gain en factorisation car les termes sont ordonnés par leur fréquence. Une distribution uniforme rend le *Trie* plus large, car plus de termes sont utilisés d'où plus de possibilité de combinaisons de souscriptions. La distribution n'a pas le même impact sur la taille du *POT* à cause du compactage des chemins.

Pour ce qui est du nombre de nœuds visités, la distribution des occurrences des termes a un impact important durant le traitement d'un item. Le nombre de nœuds visités diminue pour tous les index en passant d'une distribution réelle, à uniforme puis inversée. Dans *CIL* et *RIL*, le nombre de listes de souscriptions à traiter, ainsi que leur taille, diminuent car la probabilité d'avoir les termes de l'item dans l'index diminue. Concernant le *ROT* et le *POT*, avec une distribution inversée, peu de souscriptions correspondent aux termes fréquents de l'item, ce qui a pour conséquence d'arrêter rapidement le parcours car les termes ne sont pas présents au début du *Trie*.

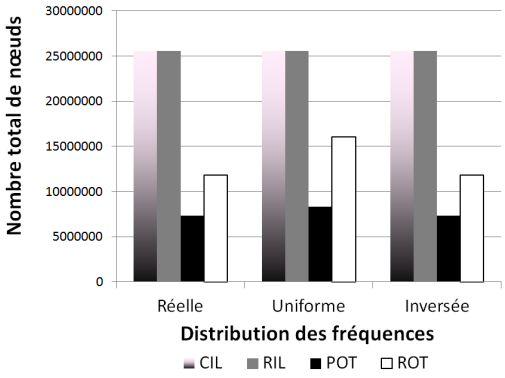

Figure 4.12 – Impact des fréquences des termes sur le nombre de nœuds dans les index

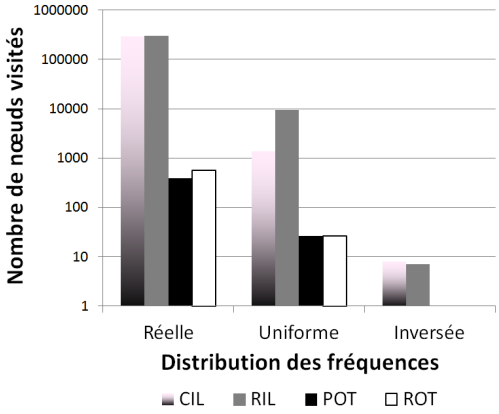

Figure 4.13 – Impact des fréquences des termes sur le nombre de nœuds visités

ROT vs. POT en faisant varier la taille du vocabulaire

ROT sera exclu des expériences de la section 5.5.4 à cause de ses besoins en espace mémoire et temps de traitement qui ne permettent pas le passage à l'échelle. Pour mieux comprendre ce comportement, nous avons étudié la morphologie des *Tries* (*ROT* et *POT*) pour différentes tailles du vocabulaire. Comme le montre la figure 4.14, le nombre de nœuds dans les index augmente linéairement avec la taille du vocabulaire. Cependant la pente est plus grande pour *ROT* qui consomme plus d'espace mémoire, illustrant que *ROT* ne permet pas le passage à l'échelle en nombre de souscriptions indexées.

Regardant le nombre de nœuds visités dans les *Tries* pour traiter un item, la figure 4.15 montre que ce nombre augmente exponentiellement pour les deux structures. En fait, le nombre de nœuds augmente avec la taille du vocabulaire, d'où plus de nœuds à visiter pendant le traitement d'un item.

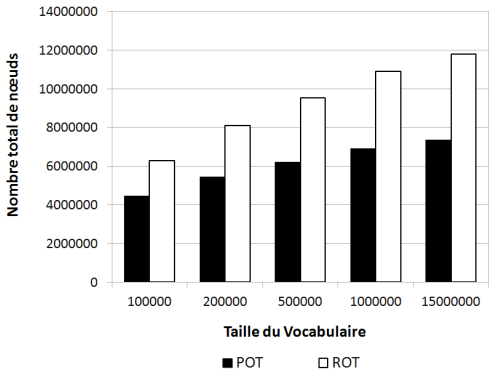

Figure 4.14 – Nombre de nœuds dans les *Tries* pour différentes tailles du vocabulaire

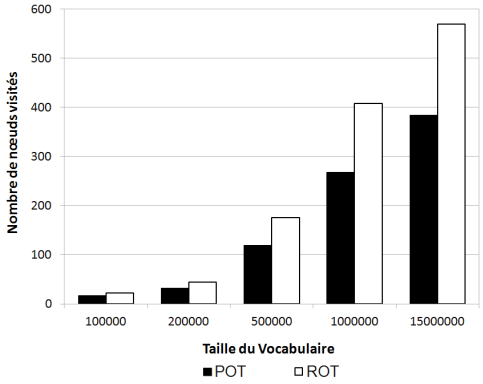

Figure 4.15 – Nombre de nœuds visités dans les *Tries* pour différentes tailles du vocabulaire

4.5.4 Evaluation des performances

Nous sommes intéressés dans cette section à comparer les performances de passage à l'échelle de *CIL*, *RIL* et *POT*. Nous étudierons leurs performances en espace mémoire et temps de traitement en faisant varier la taille du vocabulaire des souscriptions et le nombre de souscriptions indexées.

Espace mémoire

La figure 4.16 montre, en fonction de la taille du vocabulaire, l'évolution de l'espace mémoire utilisé par les trois index pour indexer 10 millions de souscriptions. En prenant des vocabulaires d'items $\mathcal{V}_I$ de taille comprise entre $100K$ et $1,5M$ termes, nous générons des souscriptions dont la taille du vocabulaire $\mathcal{V}_S$ varie de 87 839 à 471 324 termes.

En général, le *CIL* et le *RIL* utilisent un tiers de l'espace mémoire utilisé par le *POT*. L'espace mémoire utilisé par les index augmente avec la taille du vocabulaire. Mais le *POT* est plus sensible à la taille du vocabulaire, son espace mémoire augmente de 710 à 925 Mo (30%) quand $\mathcal{V}_S$ passe de 872 839 à 471 324 termes. Ceux du *CIL* et du *RIL* augmentent seulement de 10%. Avec de larges vocabulaires, les possibilités de combinaisons de souscriptions sont plus grandes, d'où une factorisation moins importante et un nombre de nœuds qui augmente dans le *POT*. Pour le *CIL* et le *RIL*, la taille du vocabulaire a un impact seulement sur la taille du dictionnaire et la majorité de l'espace mémoire est consommée par les listes des souscriptions dont la taille est fixe ($|s|_{moy} \times |\mathcal{S}|$).

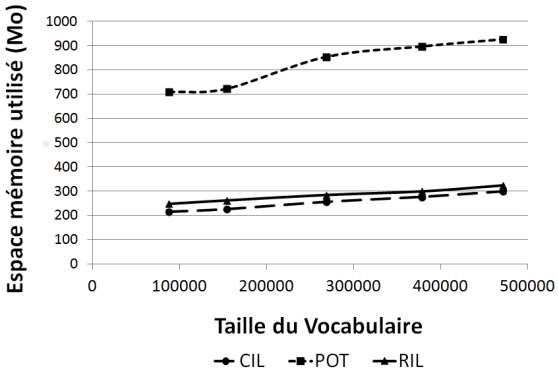

Figure 4.16 – Espace mémoire en fonction de la taille du vocabulaire

La figure 4.17 illustre l'espace mémoire consommé par les index pour $\mathcal{V}_I$ (resp. $\mathcal{V}_S$) avec $1,5M$ (resp. $1,2M$) termes en faisant varier le nombre de souscriptions de 5 à 100 millions. L'espace mémoire consommé par le *CIL* et le *RIL* augmente linéairement avec le nombre de souscriptions. Cela est dû à la taille des listes de souscriptions qui augmente. L'espace mémoire de *POT* augmente également linéairement alors que nous nous attendions à une augmentation sous-linéaire grâce à la factorisation. La raison est l'apparition de nouveaux nœuds correspondant à des combinaisons de souscriptions qui n'existaient pas. Elles sont assez nombreuses avec ce large vocabulaire. Pour indexer 100 millions de souscriptions, le *POT* utilise un espace mémoire (9,2 Go) quatre fois plus grand que celui utilisé par le *CIL* et le *RIL* (2,32 Go), malgré son gain en factorisation (voir section 4.5.3). À noter que l'espace mémoire consommé par un nœud dans *POT* dans notre implantation est en moyenne de 128 octets tandis qu'il est de 4 octets dans le *CIL* et le *RIL*.

Temps de traitement

La figure 4.18 montre que pour 10 millions de souscriptions indexées, le *RIL* et le *POT* nécessitent moins de temps pour traiter un item que le *CIL* pour toutes les tailles du vocabulaire. Par exemple pour $|\mathcal{V}_S|$ égal à $378K$, le temps de traitement pour le *CIL* est de $6,21$ ms tandis qu'il est seulement de $0,89$ ms pour le *RIL* et de $0,94$ ms pour le *ROT*. En effet, pour traiter un item dans le *CIL*, il faut parcourir les listes des sous-

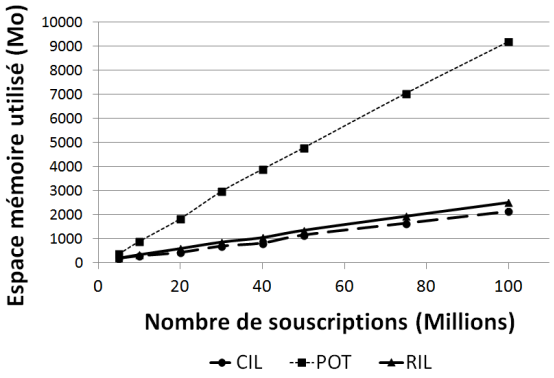

Figure 4.17 – Espace mémoire en fonction du nombre de souscriptions

criptions et décrémenter leur valeur dans le compteur. Alors que dans le *RIL*, les listes des souscriptions sont plus courtes que celles du *CIL*. De son côté, le *POT* profite de sa structure arborescente pour limiter l'espace de recherche des souscriptions dès qu'un terme de l'item n'apparaît pas dans le *Trie* (*pruning*).

Ainsi, la taille du vocabulaire $\mathcal{V}_S$ influe sur le temps de traitement des items. Dans le *CIL*, le temps augmente d'une manière convergente ; en effet, avec un large $\mathcal{V}_S$ nous observons plus de listes de souscriptions à traiter, mais de longueurs plus courtes. Pour le *RIL*, le temps de traitement augmente linéairement avec $\mathcal{V}_S$: pour de larges vocabulaires, la probabilité de trouver les termes de l'item dans le dictionnaire est plus grande, d'où le nombre de listes à traiter qui devient plus conséquent. Dans le *POT*, le temps de traitement augmente exponentiellement avec $\mathcal{V}_S$: moins de factorisation possible et plus de chemins à traverser pour traiter un item.

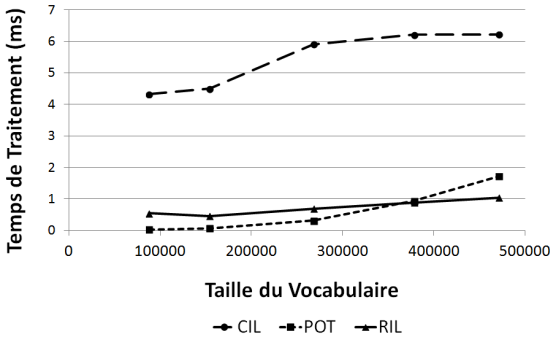

Figure 4.18 – Temps de traitement en fonction de la taille du vocabulaire

Pour finir, la figure 4.19 illustre le temps de traitement pour $\mathcal{V}_I$ égal à $1,5$M termes en faisant varier le nombre de souscriptions dans l'index de 5 à 100 millions. Pour les trois index, le temps de traitement augmente linéairement avec le nombre de souscriptions. Les listes des souscriptions dans le *CIL* et le *RIL* augmentent linéairement avec le nombre

de souscriptions indexées $|\mathcal{S}|$. Dans le *POT*, avoir un grand nombre de souscriptions augmente le nombre de combinaisons possibles des termes, ce qui augmente le nombre de chemins à parcourir.

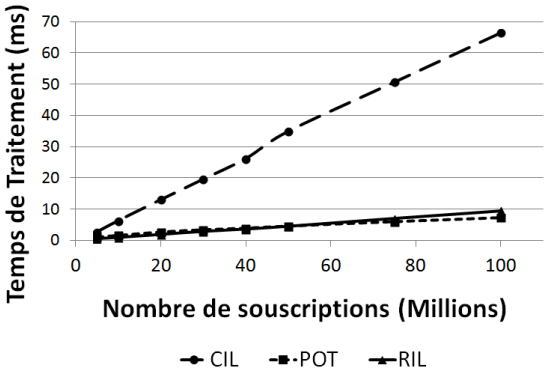

Figure 4.19 – Temps de traitement en fonction du nombre de souscriptions

Analyse du temps de traitement

Les résultats précédents montrent que *POT* est beaucoup plus rapide que *CIL* pour le traitement d'un item. Pour expliquer cette ordre de grandeur entre leur temps de traitement, nous avons mesuré le temps nécessaire pour exécuter les différentes instructions de leur algorithme de traitement. Le tableau 4.5 (resp. tableau 4.6) illustre le pourcentage de temps des instructions de *POT* (resp. *CIL*). Ces tableaux montrent les résultats moyens des expériences faites pour plusieurs jeux de souscriptions (5, 10 et 25 millions de souscriptions).

Instructions	% du temps de traitement
Trier les termes	1%
Rechercher les nœuds	89%
Appel récursif	10%

Table 4.5 – Temps de traitement pour POT

Le tableau 4.5 montre que 89% du temps de traitement d'un item dans *POT* est passé à chercher et tester l'existence des nœuds suivants à parcourir et l'étape de tri des identificateurs des termes ne consomme pas plus de 1% du temps de traitement. Pour traiter un item dans *CIL* (voir tableau 4.6), seulement 10% du temps est passé pour parcourir les listes des souscriptions et à peu près 50% du temps pour décrémenter le compteur cela est dû au besoin de traiter toutes les souscriptions des listes des termes de l'item (un de désavantages de l'algorithme basé sur le comptage). Une remarque importante est que 40% du temps de traitement dans *CIL* est utilisé pour copier le compteur, alors que

dans *RIL* cette instruction n'existe pas d'où la différence en temps de traitement entre les deux index.

Instructions	% du temps de traitement
Copie du compteur	40%
Parcourir les listes des souscriptions	10%
Décrémenter le compteur	50%

Table 4.6 – Temps de traitement pour CIL

Temps d'indexation

Finalement, nous mesurons le temps moyen d'indexation de 10 millions de souscriptions générées en utilisant un vocabulaire $\mathcal{V}_I$ de $1,5M$ termes. La construction des index basés sur les listes inverses est plus rapide que celle basée sur les *Tries*, avec seulement $0,7$ μs (resp. $1,0$ μs) pour insérer une souscription dans *RIL* (resp. dans *CIL*) tandis que pour *POT* elle est égale à 17 μs.

L'insertion d'une nouvelle souscription dans le *POT* demande des modifications dans la structure du *Trie* et dans le type de ses nœuds (ajouter une liste des identificateurs des souscriptions, un fils, etc.). Contrairement au *CIL*, l'insertion demande seulement l'insertion de l'identificateur de la souscription dans les listes de ses termes, dans le *RIL* l'indexation est plus simple puisque la souscription est ajoutée dans une seule liste.

Le tableau 4.7 résume les performances des structures de données. Le *CIL* ne consomme pas beaucoup d'espace mémoire et présente une indexation très rapide. Cependant il est est coûteux en temps de traitement. De même, le *RIL* ne consomme pas beaucoup d'espace mémoire mais nécessite plus de temps pour l'insertion. Il est cependant plus efficace en temps de traitement. Le *POT* qui utilise beaucoup d'espace mémoire, est très efficace en temps de traitement mais l'insertion dans celui-ci est plus coûteuse.

	Espace mémoire	Temps de traitement	Temps d'indexation
CIL	++	−−	++
RIL	++	+	+
POT	−	++	−

Table 4.7 – Tableau comparatif des index

Résumé

Grâce aux expériences que nous avons effectuées, nous avons pu mettre en valeur les caractéristiques de nos trois index en fonction de plusieurs paramètres.

L'étude morphologique de nos index montre un gain important de factorisation en nombre de nœuds en passant des listes inverses aux *Tries*. Comme nous avons vu dans la deuxième partie de nos expériences, ce gain est en fait théorique, car nos implantations donnent des résultats contraires, à cause de la taille en espace mémoire d'un nœud dans les trois index : *CIL* (vs. *ROT*) nécessite le moins (vs. plus) d'espace mémoire tant que *ROT* (vs. *CIL*) offert meilleure (vs. pire) performance en temps de traitement d'un item. *RIL* semble être un bon compromis pour le passage à l'échelle, aussi bien au niveau de la taille du vocabulaire que du nombre de souscriptions (et peut même donner de meilleures performances que *POT* avec de larges vocabulaires). Toutefois, le *POT* apporte de meilleures performances pour des vocabulaires de tailles raisonnables (300K), mais reste très gourmand en espace mémoire à cause de la taille des nœuds. Le *CIL* est une solution peut coûteuse en mémoire, mais à cause de son compteur et de la taille des listes inverses c'est une solution plus coûteuse en temps de traitement.

Plus précisement, pour les paramètres principaux des systèmes de syndication Web, les trois index possèdent les comportements suivants :

i) la factorisation est plus importante dans *ROT* pour de petites souscriptions ; regardant la réalité des requêtes sur le Web des petites souscriptions sont espérées [Vouzoukidou, 2010]. En plus, cette factorisation n'est pas influencée par l'ordre de termes utilisé dans *ROT* ;

ii) bien qu'elle n'affecte pas le nombre de nœuds créés dans les trois index, la distribution des occurrences des termes influe sur le nombre de nœuds visités pour le traitement d'un item. Plus particulièrement, pour une distribution réelle où les termes dans les souscriptions respectent leur nombre d'occurrences dans les items, le nombre de nœuds visités dans *CIL* est beaucoup plus important que celui dans *ROT* surtout pour les termes les plus fréquents. Quand la distribution des fréquences respecte celle de notre jeu réel d'items [Hmedeh et al., 2011b], le nombre de nœuds visités dans *CIL* est en moyenne trois fois plus que celui dans *ROT*. Bien sûr, quand la distribution de nombre d'occurrences respecte la distribution inverse, le nombre de nœuds visités est plus petit pour les deux index ;

iii) pour les trois index, plus le vocabulaire est large, plus on utilise d'espace mémoire. La taille du vocabulaire influe aussi sur le temps de traitement : pour de petits vocabulaires, *POT* est plus performant que *RIL* (resp. *CIL*) d'un ordre (resp. deux ordres) de grandeur ; mais pour de larges vocabulaires utilisés dans les systèmes de syndication Web, où le temps de traitement dans *CIL* converge, *RIL* est plus rapide que *POT* pour lequel le temps de traitement augmente exponentiellement ;

iv) finalement, l'espace mémoire et le temps de traitement augmentent linéairement dans les trois index en fonction du nombre de souscriptions indexées.

4.5.5 Satisfaction totale vs Satisfaction partielle

Comme nous avons déjà vu dans les sections précédentes, le temps de traitement dans *CIL* pour une satisfaction totale des souscriptions est beaucoup plus grand que celui dans les autres structures, principalement à cause du temps du copie du compteur. Nous avons étendu cependant cet index dans la section 4.4 pour prendre en considération la satisfaction partielle des souscriptions. Dans cette section, nous allons étudier le comportement des solutions proposées (CIL_p et CIL_{pu}), et puis les comparer aux autres index (POT et RIL) en appliquant l'approche de la décomposition des souscriptions en sous-ensembles qui satisfont la souscription.

Avant d'évaluer la performance de nos index en espace mémoire et temps de traitement pour la satisfaction partielle, nous allons étudier le taux de sous-souscriptions générées par décomposition pour un seuil donné. Les expériences sont faites pour différents ensembles de souscriptions avec différentes tailles $|S|$ et une distribution réelle pour la taille des souscriptions et les fréquences des termes. Comme le montrent les résultats du tableau 4.8, le nombre de souscriptions générées diminue avec le seuil car le nombre de combinaisons de sous-ensembles de termes qui satisfont de petits seuils est plus grand que celui pour un seuil élevé. Dans les expériences suivantes, le seuil de satisfaction par défaut est fixé à 75% et le nombre de souscriptions indexées est 10 millions.

Seuil	Taux souscriptions générées/souscriptions soumises
50%	$2,5$
60%	$2,22$
70%	$1,75$
80%	$1,33$
90%	$1,08$

Table 4.8 – Souscriptions générées par rapport au seuil de satisfaction

Espace mémoire

Les deux structures CIL_p et CIL_{pu} ont été implantées et comparées à *CIL* (où la satisfaction totale des souscriptions est considérée). La seule différence entre l'implantation de *CIL* et de ses variantes pour la satisfaction partielle est la structure des listes des souscriptions où nous avons ajouté un simple tableau de type *byte* (octet) pour sauvegarder le poids (importance) du terme pour chaque souscription (le poids d'un terme $\varrho(t_k, s) * 100 \leq 100$, alors un octet est suffisant pour le coder). L'espace mémoire utilisé en faisant varier le nombre de souscriptions indexées augmente linéairement avec le nombre de souscriptions dans les trois index (voir figure 4.20). L'espace mémoire est presque le même pour un petit nombre de souscriptions, et la différence devient plus importante avec le nombre de souscriptions. En fait, c'est l'espace mémoire consommé par les tableaux des poids des termes qui augmente avec le nombre de souscriptions. Il

est aussi important de noter que cet espace n'est pas assez significatif : pour 10 millions de souscriptions il ne dépasse pas les 20 Mo (en moyenne 2,2 entrées par souscription sont ajoutées).

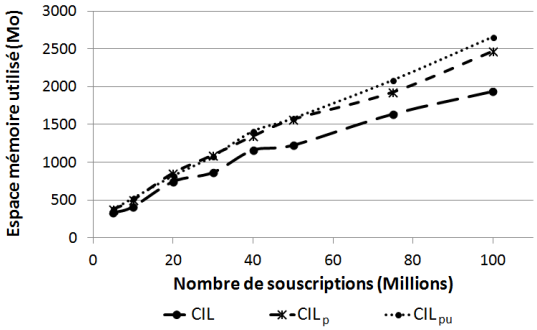

Figure 4.20 – Espace mémoire pour le CIL et ses variantes pour la satisfaction partielle en fonction du nombre de souscriptions

L'espace mémoire consommé par CIL_p et CIL_{pu} dépend de l'espace mémoire consommé par les tableaux des poids des termes. Le nombre des entrées ajoutées pour les poids des termes dépend des tailles des souscriptions. Pour étudier l'impact de la taille des souscriptions sur l'espace mémoire, nous avons fait des expériences avec 10 millions de souscriptions pour différentes tailles des souscriptions ($|s| \in \{3, 6, 9, 12, 24, 36\}$). Les résultats présentés dans la figure 4.21, montrent que l'espace mémoire dans les index augmente linéairement avec la taille des souscriptions (plus d'entrées à ajouter pour de larges souscriptions). La taille des listes des souscriptions dans CIL_p et sa variante est plus grande que celle dans CIL. La différence en espace mémoire des trois index n'est pas significative pour de petites souscriptions ($|s| = 3$) environ 100 Mo.

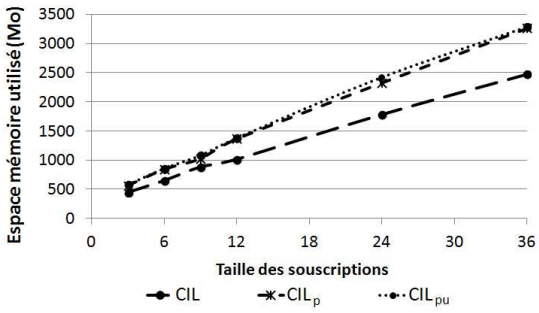

Figure 4.21 – Espace mémoire pour le CIL et ses variantes pour la satisfaction partielle pour différentes tailles de souscriptions

La figure 4.22 montre l'espace mémoire utilisé par nos trois index (CIL, RIL et POT) avec l'approche de décomposition des souscriptions pour une satisfaction partielle et celui utilisé par CIL_p, nous gardons en abscisse le nombre initial de souscriptions soumises. L'espace mémoire utilisé augmente linéairement avec le nombre de souscriptions indexées

$|S|$. Comparant l'espace mémoire utilisé avec celui d'une notification avec une satisfaction totale (section 4.5.4), en moyenne il augmente de 52% pour les trois index (58% pour *CIL*, 48% pour *RIL* et 49% pour *POT*). Mais comparant, les trois index entre eux nous remarquons le même comportement en espace consommé. Contrairement aux résultats présentés dans la figure 4.20, CIL_p consomme moins d'espace mémoire que *CIL* cela est dû au nombre effectif des souscriptions indexées dans *CIL* qui est plus grand que celui dans CIL_p en utilisant l'approche de décomposition des souscriptions.

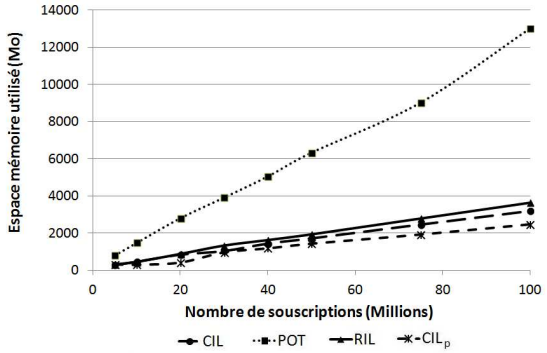

Figure 4.22 – Espace mémoire pour la satisfaction partielle en fonction du nombre de souscriptions

Nous avons vu au début de cette section que le nombre de souscriptions générées dépend du seuil de satisfaction (voir tableau 4.8). Nous avons aussi étudié son impact sur l'espace mémoire utilisé par les trois index avec l'approche de décomposition des souscriptions. figure 4.23 montre l'espace mémoire pour différentes valeurs de seuil de satisfaction (50, 60, 70, 80 et 90). L'espace mémoire utilisé diminue avec le seuil dans les trois index, car le nombre de souscriptions indexées diminue. Dans CIL_p, l'espace mémoire est indépendant du seuil, chaque souscription est indexée une seule fois quelle que soit la valeur du seuil.

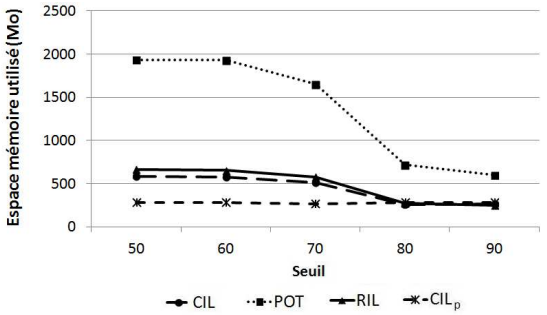

Figure 4.23 – Espace mémoire pour la satisfaction partielle pour différents seuils de satisfaction

Temps de traitement

Prendre en considération la satisfaction partielle des souscriptions dans notre système, permettra de gagner en temps de traitement d'un item. figure 4.24 illustre le temps de traitement dans *CIL*, CIL_p et CIL_{pu}. Le temps augmente linéairement avec le nombre de souscriptions dans les trois index. Le temps pour traiter un item dans CIL_p est légèrement inférieur à celui dans *CIL*. En effet, dans CIL_p nous n'avons plus besoin de faire une copie du compteur, et en plus le gain en temps augmente avec le nombre des souscriptions (*c.-à-d.*, la taille du compteur). Tandis que dans CIL_{pu}, il faut toujours faire une copie du compteur des seuils de satisfaction de chaque souscription comme dans le *CIL*. En plus, chercher le poids du terme pour le décrémenter est plus coûteux que simplement décrémenter le compteur de 1 dans *CIL*. Ce qui rend le temps de traitement dans CIL_{pu} plus élevé que celui dans *CIL*. Mais cette augmentation ne dépasse pas les 25% pour 100 millions de souscriptions.

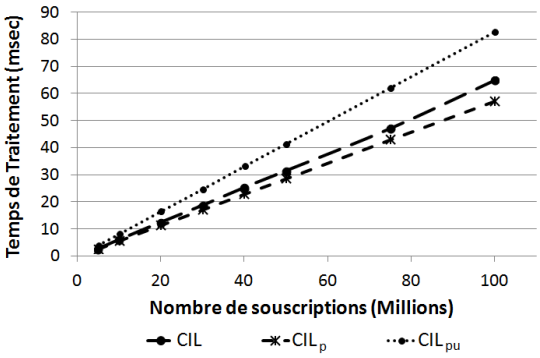

Figure 4.24 – Temps de traitement pour le CIL et ses variantes pour la satisfaction partielle en fonction du nombre de souscriptions

Avec la taille des souscriptions indexées, le temps de traitement augmente sous linéairement (voir figure 4.25). Plus les souscriptions sont larges, plus le nombre des entrées à visiter est grand, mais comparant aux réusltats précédents, la taille des souscriptions n'impacte pas la taille du compteur (*c.-à-d.*, temps du copie). La différence en temps de traitement entre *CIL* et CIL_{pu} augmente légèrement avec le nombre de souscriptions (le nombre des entrées ajoutées).

Figure 4.26 montre le temps de traitement dans nos index (*CIL*, *RIL* et *POT*) avec l'approche de décomposition des souscriptions, pour différents nombres de nombre de souscriptions indexées $|S|$. Le temps de traitement augmente linéairement avec $|S|$, et les trois index ont le même comportement qu'avec une satisfaction totale (voir section 4.5.4). Au contraire, la différence entre *CIL* et CIL_p devient plus importante. Malgré un nombre effectif de souscriptions indexées dans *RIL* et *POT* plus grand que dans CIL_p, ces index restent plus rapide que CIL_p d'un ordre de grandeur. À noter, que le temps de traitement dans *POT* avec une satisfaction partielle est presque le même avec une satisfaction totale (voir figure 4.19 et 4.26). Pour *RIL*, le temps augmente de 75% en

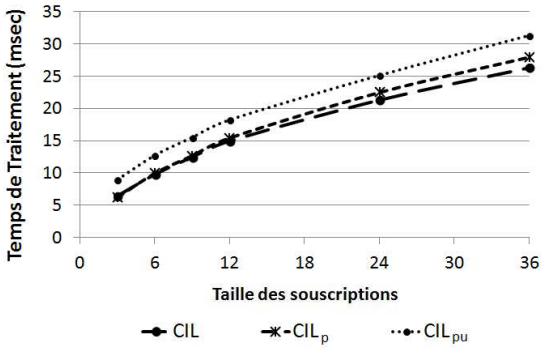

Figure 4.25 – Temps de traitement pour le CIL et ses variantes pour la satisfaction partielle pour différentes tailles de souscriptions

le comparant avec le temps passé pour une satisfaction totale, avec un large nombre de souscriptions pour le même vocabulaire (la décomposition des souscriptions augmente le nombre de souscriptions pour le même ensemble de termes d'où les listes des souscriptions deviennent plus larges et donc plus de temps pour les parcourir). Dans *CIL*, le temps augmente seulement de 21% (un compteur plus large à copier et les listes des souscriptions plus larges).

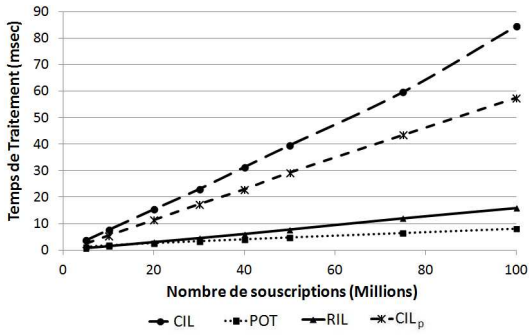

Figure 4.26 – Temps de traitement pour la satisfaction partielle en fonction du nombre de souscriptions

Nous observons également que le seuil de satisfaction impacte effectivement le temps de traitement dans les index basés sur les listes inverses (*CIL* et *RIL*), tandis que celui dans *POT* est légèrement impacté, et que le seuil n'a pas d'influence sur le temps dans CIL_p (voir figure 4.27). Dans les index basés sur les listes inverses, le nombre de souscriptions indexées est plus grand pour de petites valeurs pour le seuil de satisfaction d'où des listes de souscriptions larges, et donc un temps de traitement qui diminue avec le seuil. Le temps de traitement dans *POT* dépend essentiellement du nombre de nœuds visités et non du nombre de souscriptions indexées. Le nombre de nœuds visités dépend de la taille du vocabulaire qui ne change pas avec la décomposition des souscriptions. Le nombre de

souscriptions augmente pour le même vocabulaire. À noter que les souscriptions générées sont plus courtes d'où le *Trie* est moins profond.

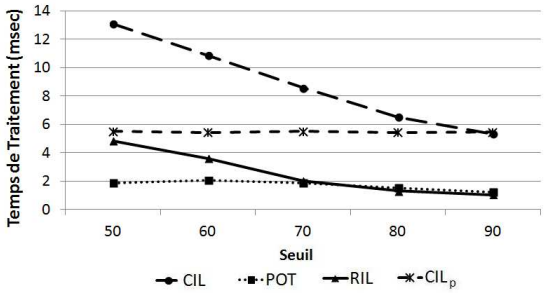

Figure 4.27 – Temps de traitement pour la satisfaction partielle pour différents seuils de satisfaction

Temps d'indexation

Pour ajouter une souscription dans CIL_p et sa variante, il faut en plus des instructions à faire dans *CIL*, ajouter le poids du terme dans la liste des souscriptions ce qui augmente le temps d'indexation de 20% ($0,85\ \mu s$). Ce temps augmente avec la taille des souscriptions. Figure 4.28 montre le temps d'indexation pour différentes tailles de souscriptions. La différence en temps d'indexation entre *CIL* et CIL_p et sa variante est presque constante ($\sim 20\%$).

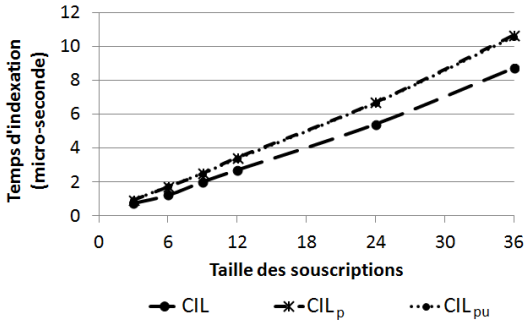

Figure 4.28 – Temps d'indexation pour la satisfaction partielle en fonction de la taille des souscriptions

Avec l'approche de décomposition des souscriptions, le temps d'indexation augmente. Pour indexer une souscription, nous aurons besoin de $6,37\ \mu s$ (resp. $6,28\ \mu s$) dans *CIL* (resp. *RIL*) et $8,7\ \mu s$ dans *POT*. Le temps moyen pour générer les sous-souscriptions est égale à $5,4\ \mu s$. D'où le temps d'indexation dépend du temps de génération des souscriptions qui satisfont le seuil. Nous avons fait des expériences pour étudier l'impact du seuil sur le temps d'indexation. Figure 4.29 montre que le temps d'indexation diminue

avec le seuil de satisfaction. Pour de petites valeurs du seuil, le nombre de souscriptions générées est plus grand d'où plus de temps pour les générer et les ajouter aux index. Le temps d'indexation dans CIL_p est indépendant du seuil.

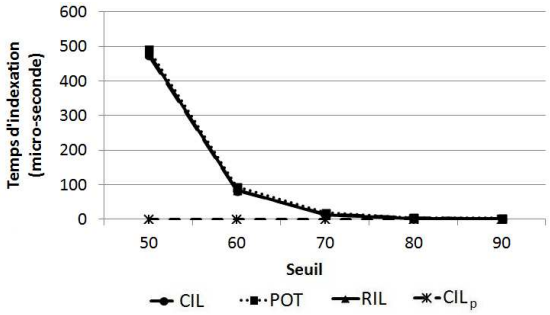

Figure 4.29 – Temps d'indexation pour la satisfaction partielle pour différents seuils de satisfaction

Résumé

Globalement, CIL_p a besoin moins de temps pour traiter un item que CIL mais ses besoins en espace mémoire sont plus importants. Même avec l'introduction de la satisfaction partielle dans nos index, nous avons pu comparer leurs performances à celles d'une satisfaction totale présentées dans la section précédente (section 4.5). Les principales conclusions des expériences de cette section sont les suivantes :

i) la prise en considération de la satisfaction partielle des souscriptions dans les index basés sur les listes inverses (CIL) permet un gain en temps de traitement en évitant la copie du compteur ;
ii) l'espace mémoire consommé par les tableaux pour les poids des termes sauvegardés dans les listes des souscriptions de CIL_p n'est pas significatif ;
iii) l'approche par décomposition des souscriptions augmente l'espace mémoire consommé par les index, qui dépasse celui de CIL_p ;
iv) malgré le nombre effectif des souscriptions indexées dans RIL et POT avec la décomposition des souscriptions, ces index restent plus rapide que CIL_p et le temps de traitement dans POT est presque le même que celui avec une satisfaction totale des souscriptions ;
v) les performances du CIL_p sont indépendantes du seuil de satisfaction, tandis que le temps d'indexation, l'espace mémoire utilisé et le temps de traitement sont plus importants pour de petites valeurs de seuil dans les trois index avec l'approche de décomposition des souscriptions.

4.6 Conclusion

Nous avons donc présenté trois structures d'indexation de souscriptions dans les systèmes Pub/Sub et leur modèle analytique pour estimer leur taille et le temps nécessaire pour traiter un item. Nous avons également proposé des approches pour nos index qui permettent la recherche des souscriptions qui sont partiellement satisfaites par les items publiés. Nous avons aussi présenté une étude comparative approfondie des comportements et performances des index proposés pour la notification des souscriptions dans les systèmes Pub/Sub. Nous avons comparé la morphologie des index basés sur les listes inverses aux structures arborescentes (*Tries*). Un gain important en nombre de nœuds dans les *Tries* dû à la factorisation entre les souscriptions qui n'est pas affectée par l'ordre des termes, ni la distribution des fréquences des termes dans les souscriptions. Le *Trie*, malgré ce gain, consomme beaucoup d'espace mémoire en comparaison avec les listes inverses. Il est cependant toujours plus rapide que les listes inverses pour le passage à l'échelle en nombre de souscriptions indexées. À noter que *RIL* (listes inverses basées sur le mot clé dans la souscription) peut être un compromis entre le *Trie* et les listes inverses avec l'algorithme de comptage pour l'espace mémoire et le temps de Traitement. L'introduction de la satisfaction partielle n'impacte pas les performances en temps de traitement dans le *Trie*, malgré le nombre effectif des souscriptions indexées avec l'approche par décomposition des souscriptions. Reste à noter que le temps d'indexation, l'espace mémoire utilisé et le temps de traitement dans nos index avec cette approche sont impactés par le seuil de satisfaction qui n'a aucune influence sur CIL_p la variante de *CIL* pour la satisfaction partielle. Cette dernière permet un gain en temps de traitement en évitant la copie du compteur et ne consomme pas beaucoup d'espace mémoire en comparaison avec le *CIL*.

Avec le taux de publication élevé des flux sur le Web, l'utilisateur peut être submergé par la quantité d'information reçue. Dans le chapitre suivant, nous proposerons un filtrage des items qui satisfont une souscription pour réduire l'ensemble d'items notifiés à l'utilisateur. De fait, nous nous focaliserons sur la diversité et la nouveauté de l'information publiée dans les items des flux RSS.

Chapitre 5

Filtrage par nouveauté et diversité

5.1 Introduction

Dans le chapitre précédent, nous avons proposé un système de Publication/Souscription pour la syndication Web qui permet aux utilisateurs d'exprimer leurs intérêts par des souscriptions sous forme d'ensembles de mots-clés et de recevoir les items qui satisfont ces souscriptions. Cependant, le grand nombre de flux sur le Web et leur taux de publication élevé [Hmedeh et al., 2011b], font que le nombre d'items notifiés à l'utilisateur pour une souscription donnée reste très important et s'apparente à du « *flooding* ». Dans ce chapitre, nous nous intéressons à des systèmes de filtrage qui réduiront l'ensemble d'items notifiés par souscription et plus particulièrement, nous nous focaliserons sur la diversité et la nouveauté de l'information. À noter que le filtrage que nous proposerons dans ce chapitre est indépendant du système d'indexation de souscriptions pour le *matching* des items et donc n'importe lequel des index étudiés dans le chapitre 4 peut être employé pour l'indexation des souscriptions. Nous proposerons le système de la figure 5.1 où après vérification de la satisfaction des souscriptions par un item, cet item sera ensuite filtré par la nouveauté et la diversité de l'information qu'il contient afin de décider s'il sera envoyé à l'utilisateur.

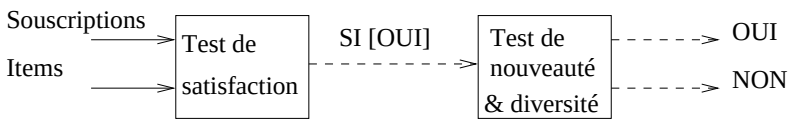

FIGURE 5.1 – Système de notification à deux phases

5.2 Nouveauté et diversité

Le filtrage des items est effectué par suppression de l'information redondante (*c.-à-d.*, nous nous intéressons à la *Nouveauté*) et par prise en compte de la diversité de l'information dans les items délivrés à l'utilisateur (*c.-à-d.*, *Diversité*). L'objectif de ce filtrage est d'envoyer à l'utilisateur des informations variées non-redondantes et couvrant toutes les actualités par un nombre minimal d'items satisfaisant l'utilisateur au niveau de la qualité (pertinence de l'information) et la quantité (ne pas être submergé). Cette section présente les définitions de nouveauté et diversité ainsi qu'un exemple de filtrage par nouveauté et diversité.

5.2.1 Nouveauté

La nouveauté de l'information a été étudiée dans le contexte de la recherche d'information dans le but d'éviter l'envoi d'information redondante à l'utilisateur. Cela peut être réalisé en envoyant seulement les documents qui contiennent de l'information qui n'a pas été délivrée dans les documents précédents [Zhang et al., 2002, Clarke et al., 2008]. Cela correspond à une relation de couverture entre les documents : un document est considéré comme nouveau s'il n'est pas couvert par un autre. Par exemple, un paragraphe p d'un document d ($p \subset d$) n'est pas considéré comme nouveau par rapport à d car il ne contient pas une nouvelle information non contenue dans d, par contre d est bien considéré comme nouveau par rapport à p. Le problème de la nouveauté de l'information a également été étudié pour détecter les duplicatas de pages Web, les plagiats... Plusieurs techniques ont été utilisées dans la littérature pour mesurer la distance entre deux documents afin d'estimer leur nouveauté : Cosinus, intersection d'ensemble de mots-clés, Jaccard...

5.2.2 Diversité

Le but de la diversité est de réduire le nombre de documents transmis contenant de l'information déjà connue (d'après l'ensemble des informations déjà reçues) par l'utilisateur. Les moteurs de recherche ont pour objectif de ne présenter à l'utilisateur dans la première page qu'un ensemble de documents couvrant un ensemble maximal de sujets qui pourraient satisfaire sa requête (surtout dans le cas de requêtes ambigües). Cette approche permet à l'utilisateur de trouver au moins un résultat qui satisfait ses intérêts. Par exemple, un utilisateur qui soumet la requête « Football 2010 » n'est pas intéressé par les nouvelles d'une seule équipe de football. D'où l'intérêt de lui présenter les nouvelles de toutes les équipes.

Dans le même objectif, la diversité a été utilisée pour afficher les résultats dans de larges bases de données. La diversité est souvent mesurée comme étant la distance moyenne entre les documents. Le problème de la diversité correspond à présenter à l'utilisateur

les documents parmi les résultats de sa requête qui sont les moins similaires entre eux [Drosou et al., 2009, Drosou and Pitoura, 2009, Pripužić et al., 2008].

Dans la littérature, le problème de la diversité est souvent présenté comme un problème de *top-k*. Soit R l'ensemble des documents qui satisfont une requête donnée, et *dist* une distance métrique. Le problème est de trouver les k documents qui favorisent la diversité (*c.-à-d.*, qui maximisent la distance moyenne entre les documents).

Définition 6 (Problème de diversité) *Soit $R = \{d_1, d_2, \ldots, d_n\}$ un ensemble de n documents, et dist une distance métrique, le problème est de trouver $R_{k_{k \leq n}} \subseteq R$ qui a la distance moyenne $D(R_k)$ maximale entre ses documents :*

$$D(R_k) = \frac{2}{|k| * (|k| - 1)} \sum_{d \in R_k} \sum_{(d' \in R_k \wedge d' \neq d)} dist(d, d')$$

Dans les applications de syndication Web, quand un nouvel item est publié le système doit décider de l'envoyer à l'utilisateur ou non. Dans ce contexte, le problème de nouveauté et de diversité ne peut pas être considéré comme un problème de *top-k* classique sur un ensemble connu d'items puisque les items déjà notifiés ne peuvent pas être retirés de R_k. Plusieurs modes de diffusion ont été utilisés pour les flux de données dans le but de limiter l'ensemble d'items à prendre en considération pour calculer la nouveauté et la diversité de nouveaux items publiés. Une fenêtre glissante basée sur le temps, sur un nombre limité d'items, ou encore un mode périodique de notification, sont utilisés dans les systèmes Pub/Sub [Drosou et al., 2009, Pripužić et al., 2008].

Dans notre contexte de flux RSS, nous nous intéressons à filtrer les items par nouveauté et diversité en se basant sur l'information déjà délivrée à l'utilisateur dans les items précédents. Ainsi, nous adaptons une fenêtre glissante basée sur le temps pour gérer l'historique des items reçus par l'utilisateur. Une fenêtre glissante sur le temps permet de retirer les items trop anciens notifiés à l'utilisateur. Nous présentons dans ce qui suit un exemple de filtrage par nouveauté et diversité, et dans la section suivante nous définissions la nouveauté et diversité des items dans les Pub/Sub avant de présenter notre solution dans la section 5.4.

5.2.3 Exemple de nouveauté et diversité

Pour mieux expliquer les définitions de nouveauté et diversité pour les items des flux RSS, nous allons présenter comment nous appliquons le filtrage des items en se basant sur l'ensemble d'items reçus par l'utilisateur. Soit s = « Football 2010 » une souscription déjà notifiée par les items suivants :
– I_1 = Football 2010 Italie
– I_2 = Football 2010 France Allemagne
Le tableau 5.1 présente un ensemble de nouveaux items publiés et les résultats de leur filtrage par nouveauté et diversité. Le premier item est inclu dans I_1. Par conséquent,

il n'est pas nouveau et il ne favorise pas non plus la diversité car il est très similaire aux items déjà délivrés. Le deuxième n'est couvert par aucun items de l'historique, mais l'information contenue dans cet item est déjà délivrée par l'ensemble des items reçus, il est donc considéré comme nouveau mais ne favorise pas la diversité. Le dernier item contient de l'information nouvelle en le comparant à chaque item de l'historique, et il est considéré comme nouveau. L'information qu'il contient, n'est pas délivrée par l'information globale des items de l'historique il augmente donc la dissimilarité entre les items et favorise la diversité.

	Nouveauté	Diversité
Football 2010 France	X	X
Football 2010 France Italie	√	X
Football 2010 Italie Brésil	√	√

Table 5.1 – Exemple de nouveauté et diversité

5.3 Pub/Sub : nouveauté et diversité

Pour les items des flux RSS, le filtrage des items se base sur l'historique des items déjà notifiés pour chaque souscription. Nous ne cherchons pas à trouver les items qui contiennent de l'information nouvelle et qui sont les plus dissimilaires entre eux. En effet, l'utilisateur a déjà reçu un ensemble d'items (historique) et nous devons vérifier si le nouvel item publié contient donc de l'information utile, donc nouvelle et assez différente de l'information dans l'historique. Nos définitions de nouveauté et diversité sont basées sur l'importance (poids) des termes des items. Nous utilisons la technique de pondération utilisée dans la section 4.4 pour la satisfaction partielle des souscriptions : le *TDV* (*Term Discrimination Value*). Cette technique est basée sur l'impact d'un terme sur la densité de l'ensemble d'items (entropie) : le terme qui permet de distinguer le plus les items entre eux (plus discriminant) est plus important (pour plus de détails voir la section 4.4.1).

Dans cette section, nous présentons les définitions de nouveauté et de diversité des items et la façon de les calculer. Finalement, nous expliquons notre choix pour la fonction utilisée pour mesurer la distance entre les items.

5.3.1 Nouveauté des items

Afin de supprimer les items redondants qui ne fournissent pas de nouvelles informations à l'utilisateur, nous mesurons la similarité item-item pour vérifier si l'item fournit une nouvelle information (qui n'est pas apparue dans un ancien item). C'est une mesure de

couverture (*text overlap*) d'un item par un autre. Nous nous intéressons donc à définir un seuil de couverture, afin de définir un niveau d'intérêt que le nouvel item doit fournir.

Pour mesurer la nouveauté d'un item par rapport à un autre, nous prenons en considération l'importance de ses termes qui n'apparaissent pas dans l'autre item par rapport à son ensemble total de termes. Nous avons choisi cette mesure plutôt qu'une mesure basée sur l'intersection des ensembles comme *Jaccard* dans le but d'avoir une mesure asymétrique qui prend en considération l'ordre d'arrivée des items.

Soient deux items I et I' tel que l'intersection entre eux ($I \cap I'$) est importante par rapport au reste des termes de I qui ne fournissent pas d'information nouvelle : si I est publié avant I', I sera notifié à l'utilisateur mais si I' a été déjà notifié à l'utilisateur I ne sera pas notifié car il ne contient pas de nouvelle information par rapport à celle de I'. Ainsi, $Nouveauté(I, I') \neq Nouveauté(I', I)$.

Nous considérons également que tous les termes n'ont pas la même importance dans un item. Ainsi notre mesure de la nouveauté ne se base pas sur la seule présence de termes nouveaux, mais par la présence de termes nouveaux importants (*c.-à-d.*, dont le poids est important). La nouveauté de l'item sera donc mesurée comme le ratio de la somme des poids de ses termes qui n'apparaissent pas dans l'ancien item sur la somme des poids de tous ses termes (Définition 7).

Définition 7 (Nouveauté item-item) *Soit α un seuil de nouveauté $\in [0, 1]$ et soient I et I' deux items. I est considéré comme nouveau par rapport à I' si et seulement si :*

$$\frac{\sum_{t \in (I \setminus I \cap I')} tdv(t)}{\sum_{t' \in I} tdv(t')} \geq \alpha$$

La nouveauté d'un item par rapport à un ensemble d'items reçus par l'utilisateur (l'historique d'une souscription), est définie comme suit :

Définition 8 (Nouveauté item-historique) *Soient H un historique d'items et I un item publié. I est dit nouveau par rapport à l'historique H, si et seulement si I est considéré comme nouveau par rapport à tous les items $I' \in H$:*

$$\forall I' \in H, \frac{\sum_{t \in (I \setminus I \cap I')} tdv(t)}{\sum_{t' \in I} tdv(t')} \geq \alpha$$

5.3.2 Diversité des items

Un filtrage uniquement basé sur la nouveauté s'avère souvent insuffisant lorsqu'on souhaite recevoir (autant que possible) des items ayant des informations différentes. En effet, un item peut ne pas être redondant avec un autre de l'historique mais contenir une information qui est présentée globalement dans les items précédemment notifiés. Un item est considéré comme intéressant, si l'information qui le compose ne correspond pas aux

sujets notifiés récemment. Techniquement, cela correspond à rechercher les termes qui ne sont pas en communs entre le nouvel item et l'ensemble des items précédemment notifiés (item - ensemble d'items), afin d'éliminer les items qui ne fournissent pas des informations diverses par rapport aux précédentes.

Le but de la diversité est de filtrer les items qui n'apportent pas d'information nouvelle. Pour cela, nous mesurons l'importance d'un item favorisant la diversité de l'ensemble des items envoyés à l'utilisateur (item-historique) comme étant la variation de la distance moyenne entre les items de l'historique avec et sans ce nouvel item. Donc un item favorise la diversité d'un ensemble si son insertion dans l'historique va augmenter la distance moyenne entre les items (appelée aussi *densité de l'ensemble*).

Soit *dist* une distance métrique pour mesurer la distance entre les items (nous étudierons le choix de la distance plus tard), la diversité des items est formellement définie comme suit :

Définition 9 (Diversité des items) *Soit H un historique d'items avec $D(H)$ la distance moyenne entre ses items. Un item I favorise la diversité de H si et seulement si :*

$$D(H \cup \{I\}) > D(H)$$

avec

$$D(H) = \frac{2}{|H| * (|H| - 1)} \sum_{I \in H} \sum_{(I' \in H \wedge I' \neq I)} dist(I, I')$$

Distance entre les items

La formule précédente nous indique que pour mesurer l'importance d'un item I à favoriser la diversité, nous devons calculer la distance moyenne entre chaque paire d'items de l'historique. Pour mesurer la distance entre les items, plusieurs approches ont été utilisées dans la littérature : *Cosinus* [Zhang et al., 2002], *Euclidienne* [Drosou and Pitoura, 2012a, Pripužić et al., 2008], et *Jaccard* [Drosou and Pitoura, 2012b].

La mesure de *Jaccard* calcule la distance comme étant le rapport de l'intersection des ensembles de termes des items, sur l'ensemble (union) des termes des deux items. Cependant, cette mesure ne prend pas en considération l'importance des termes des items. La distance *Cosinus* permet de calculer la distance entre deux vecteurs de poids des termes, mais elle prend en considération seulement les termes de l'intersection entre les items. Notre objectif de filtrage par diversité est de mesurer la nouvelle information contenue dans les items. Nous cherchons par conséquent à connaître l'importance des termes qui ne sont pas en commun. La distance *Euclidienne* permet de mesurer une distance entre les items qui intègre ce critère. Pour prendre en considération la taille des items ainsi que l'importance des autres termes des items, il faut de plus une normalisation de la distance

Euclidienne par le produit de la norme *Euclidienne* des deux vecteurs. D'où le choix de la distance entre items suivant :

Définition 10 (Distance Euclidienne entre deux items) *Soient I et I' deux items, et soit tdv(t) le poids du terme t :*

$$dist(I, I') = \frac{\sqrt{\sum_{t \in (I \cup I' \setminus I \cap I')} tdv^2(t)}}{\|I\|^2 \times \|I'\|^2}$$

À noter que le poids d'un terme (tdv) est mesuré sur l'ensemble d'items et non sur l'item considéré. De plus, les termes apparaissant dans les deux items seront ignorés. La distance entre deux items est donc basée sur l'importance des termes qui apparaissent uniquement dans l'un de deux items (information non commune entre deux items).

Pour résumer, le tableau 5.2 présente une comparaison entre la mesure de nouveauté et celle de diversité. La nouveauté est une mesure asymétrique entre les items (tient compte de l'ordre d'arrivée) pour mesurer la nouvelle information. Cette dernière est mesurée entre deux items comme étant l'importance des termes qui apparaissent dans le nouvel item publié, et non présents dans l'ancien item. La diversité d'un item par rapport à un ensemble d'items est basée sur une mesure de distance symétrique entre les items. Cette distance doit prendre en considération l'importance de l'information qui n'est pas en commun entre les items à l'aide, par exemple, d'une mesure comme la distance *Euclidienne*.

Nouveauté	**Diversité**
Mesure de la similarité : Asymétrique	Mesure de la distance : Symétrique
Mesure locale : apparition de nouveaux termes importants par rapport à un autre item → nouvelle information	Mesure globale : espace vectoriel d'items, et mesure de la distance moyenne entre les vecteurs, pour déterminer la densité de l'ensemble d'items
Degré de nouveauté de I_1 par rapport à I_2 (les nouveaux termes de I_1 qui ne sont pas dans I_2)	Distance Euclidienne pour mesurer la densité de l'ensemble

Table 5.2 – Nouveauté vs. Diversité

5.4 Approche de filtrage par nouveauté et diversité

La diversité et la nouveauté des items publiés par les flux RSS permettent de filtrer un type d'information qui ne doit pas être notifié. Ainsi, nous proposons d'intégrer les deux mesures dans notre étape de filtrage, tout d'abord en mesurant la nouveauté (si un item n'est pas nouveau, nous arrêtons le processus), puis la diversité (une fois que tous les items ont été vérifiés, nous calculons la distance moyenne). Dans cette section,

nous présentons notre solution pour gérer l'historique d'items des souscriptions. Nous allons étudier comment nous stockons les items pour réduire l'espace mémoire utilisé et testerons la nouveauté et la diversité d'un nouvel item publié.

5.4.1 Séparation des processus

Pour la prise en compte de la nouveauté et de la diversité, deux solutions semblent envisageables : les évaluer en même temps que le processus de *matching* de la souscription sur l'item entrant, ou alors considérer deux processus distincts (*matching* de la souscription, puis ensuite vérification de la nouveauté/diversité).

Dans la littérature, des travaux proposent des formules pour mesurer la pertinence et la diversité des documents en une seule passe sur les documents [Carbonell and Goldstein, 1998, Angel and Koudas, 2011], car leur but est de chercher un compromis entre la pertinence et la diversité. Dans notre cas, les deux processus peuvent être dissociés, car nous ne testons la nouveauté et la diversité de l'information que pour les items satisfaisant la souscription (*matching* puis filtrage). La satisfaction totale ou bien partielle (*broad* ou bien *partial matching*) de la souscription par l'item est obligatoire. L'avantage de dissocier les deux processus est d'avoir une structure optimisée pour chacun. Surtout, nous ne faisons pas le calcul de diversité (assez coûteux) que pour des items filtrés par la satisfaction de la souscription et non pour tous les items entrants.

Dans ce but, nous proposons le système de la figure 5.2 où le processus de tester la satisfaction de l'item pour les souscriptions est séparé de celui du filtrage.

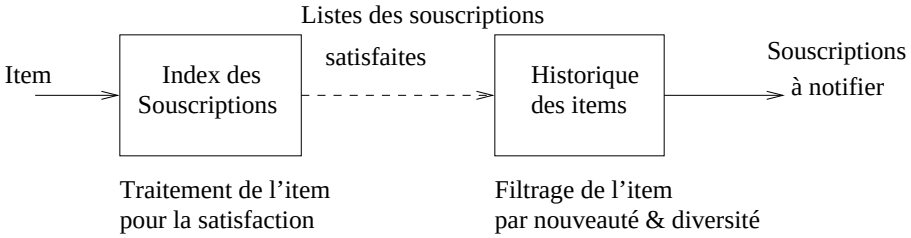

Figure 5.2 – Architecture du système

5.4.2 Historique et fenêtrage

Pour faire la comparaison d'un item filtré par la première étape (*matching*), un historique des items déjà notifiés de la souscription concernée est nécessaire. Plusieurs solutions existent pour définir l'historique : fenêtre temporelle glissante de taille fixe, fenêtre glissante avec nombre d'items fixe, etc.

Dans les systèmes basés sur des fenêtres à nombre d'items fixe (N), à l'arrivée d'un nouvel item, s'il est jugé comme nouveau et favorisant la diversité, le plus ancien item de

l'historique sera supprimé pour ajouter ce nouvel item. Quand la fenêtre glissante utilisée est basée sur le temps, seuls les items ayant été notifiés durant la période entre τ et $\tau - p$, où τ est le temps de publication de l'item et p est la durée de la fenêtre, sont pris en considération. Nous estimons qu'une solution basée sur une fenêtre glissante sur le temps est plus adéquate pour les applications Web car l'utilisateur s'intéresse aux actualités et aux nouvelles sur le Web (fraîcheur de l'information). De plus, une souscription rarement notifiée peut recevoir les N items sur une large période, ou à l'opposé en période de *burst* (pics) la fenêtre permettra d'équilibrer plus convenablement la charge.

5.4.3 Processus de filtrage

Le processus de filtrage d'un item pour un historique H donné est illustré par les instructions de l'algorithme 5. La première phase du filtrage consiste à tester la nouveauté de l'item en le comparant aux items de H. Dès qu'un item de H ne permet pas de prouver la nouveauté de I, ce dernier est rejeté, ce qui permet d'arrêter le traitement de I et de ne pas tester la diversité qui est coûteuse. Si I est nouveau, la densité de $H \cup \{I\}$ est calculée pour mesurer si I favorise la diversité de l'ensemble des items déjà notifiés (H). Si c'est le cas, I est envoyé à l'utilisateur et est ajouté à l'historique de la souscription.

Algorithme 5: Filtrage par nouveauté et diversité

Entrées : Un item I, un historique H et un seuil de nouveauté $\alpha \in [0, 1]$

```
1: pour tout I' ∈ H faire
2:    si Nouveauté(I, I') < α alors
3:       return ;
4:    finsi
5: fin pour
6: si D(H ∪ {I}) > D(H) alors
7:    Notifier I ;
8:    H ← H ∪ I ;
9: finsi
```

5.4.4 Filtrage basé sur une fenêtre glissante

Pour prendre en considération le problème de passage à l'échelle sur le Web avec un débit élevé d'items, nous avons choisi de ne pas stocker l'ensemble des items qui satisfont les souscriptions lors du *matching*. Nous ne gardons pas une fenêtre sur l'ensemble d'items qui satisfont la souscription mais sur l'ensemble d'items qui ont été notifiés à l'utilisateur après le filtrage. Quand un nouvel item est publié à un temps τ, durant la vérification de sa nouveauté par rapport aux items de l'historique de la souscription, les items qui n'ont pas été publiés dans l'intervalle de temps $[\tau - p, \tau]$ sont ignorés. Si I n'est pas nouveau ou ne favorise pas la diversité des items de l'historique, il ne sera pas stocké pour être

évalué à un autre instant de temps. Par exemple, I_3 dans l'exemple d'historique de la figure 5.3, n'a pas été envoyé à l'utilisateur et il n'a pas été pris en considération pour calculer la nouveauté et diversité de I_5, bien qu'il soit dans la fenêtre $\tau_5 - p$ (τ_5 est le temps de publication de I_5). I_3 n'est pas stocké dans l'historique. La nouveauté et la diversité de I_5 sont calculées en se basant sur l'historique $H = \{I_4\}$. À temps τ_5, I_1 et I_2 ne sont plus dans la fenêtre et sont supprimés de l'historique. À noter que pour chaque nouvelle fenêtre, après avoir supprimé les anciens items, la distance moyenne D entre les items restants doit être calculée (dans la section suivante nous expliquerons comment ce calcul sera optimisé).

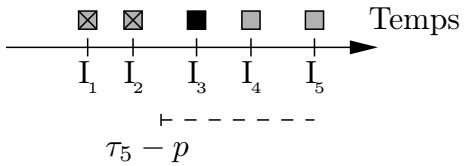

Figure 5.3 – Fenêtre glissante des items basée sur le temps

Factorisation des historiques des souscriptions

Plutôt que de garder un historique pour chaque souscription, nous fusionnons tous les historiques dans le but de réduire l'espace mémoire consommé par les items. Cela nous permettra pour chaque item notifié de n'être stocké qu'une seule fois, quel que soit le nombre d'historiques dans lesquels il est présent. Dans l'hypothèse où nous considérons une fenêtre glissante pour les historiques des souscriptions, la fusion des historiques respecte aussi cette contrainte, ce qui limite le nombre d'items à garder. Pour garder la trâce de l'historique pour chaque souscription, nous gardons pour chaque souscription des pointeurs vers les items constituant son historique.

Comme le montre la figure 5.4, nous avons un seul historique commun entre les souscriptions, chaque item est sauvegardé une seule fois et à chaque souscription est associée une liste de pointeurs vers les items de son historique. À chaque souscription correspond une liste chaînée des tableaux des pointeurs vers les items. Pour profiter de la contrainte du temps dans notre fenêtre, les items sont insérés suivant leur ordre inverse d'arrivée c'est-à-dire le premier item de l'historique de la souscription est le dernier item envoyé à l'utilisateur et les items seront traités comme une pile. Nous optimisons ainsi le temps de traitement d'un item, en arrêtant de le comparer avec les items de l'historique dès qu'un de ces items n'est pas dans la fenêtre. De plus, les items publiés consécutivement sur le Web peuvent contenir la même information et ne pas être divers. Alors en commençant la comparaison du nouvel item publié par les items les plus récents (ajoutés au début de la liste) nous augmentons les chances d'arrêter son traitement le plus tôt possible. Pour chaque item nous sauvegardons son identificateur, la liste de ses termes ainsi que son temps d'arrivée (*timestamp*).

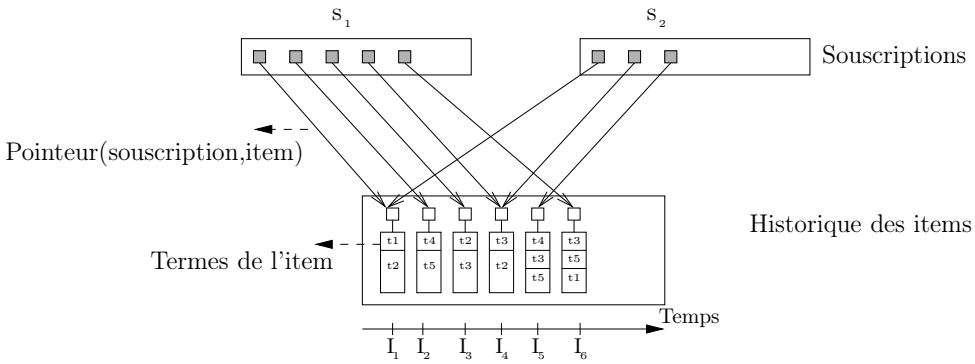

Figure 5.4 – Exemple d'un historique commun

5.4.5 Processus de filtrage basé sur une fenêtre glissante

L'algorithme 6 présente le processus de filtrage des items sur une fenêtre glissante sur le temps. En plus de l'optimisation en espace mémoire utilisé pour stocker les historiques des souscriptions, plusieurs optimisations pour améliorer le temps de traitement d'un item sont faites.

Pour commencer, il y a une grande probabilité que les souscriptions, qui sont satisfaites pour un item donné, aient un nombre important d'items en commun entre leur historique. Notre algorithme calcule la nouveauté et la distance entre le nouvel item et chaque item de l'historique une seule fois, quelque soit le nombre de souscriptions pour lesquelles il est notifié. Cette optimisation est particulièrement intéressante pour les items de grandes tailles qui satisfont un grand nombre de souscriptions, et dont le temps de calcul pour leur nouveauté et leur distance est significatif.

Pendant la vérification de la nouveauté et la diversité d'un item pour un historique H d'une souscription, au lieu de recalculer toutes les distances de chaque paire d'items restant dans l'historique et calculer la nouvelle distance moyenne entre eux, pour chaque item I dans H, nous sauvegardons la somme sum_I de ses distances avec les autres items de l'historique qui sont plus récents. sum_I est égale à :

$$sum_I = \sum_{(I' \in H \wedge \tau_{I'} > \tau_I)} dist(I, I')$$

Lors du test de la nouveauté d'un item publié I_n qui consiste à le comparer aux items de l'historique nous calculons la somme des distances sum_H. Nous obtenons la somme des distances entre les items restants dans l'historique donnée par l'équation suivante :

$$sum_H = \sum_{I \in H} sum_I = \sum_{I \in H} \sum_{(I' \in H \wedge \tau_{I'} > \tau_I)} dist(I, I')$$

Algorithme 6: Filtrage par nouveauté et diversité sur une fenêtre glissante

Entrées : Un item I, un historique H et un seuil de nouveauté $\alpha \in [0, 1]$

```
1: sum_H ← 0 ;
2: sum_I ← 0 ;
3: pour tout I' ∈ H faire
4:    si τ_I' < τ_I − p alors
5:       break ;
6:    finsi
7:    si I.getInfo(I') = null alors
8:       N ← Nouveauté(I, I')
9:       d ← dist(I, I')
10:      I.putInfo(I',N,d)
11:   sinon
12:      N ← I.getInfo(I').N
13:      d ← I.getInfo(I').d
14:   finsi
15:   si N < α alors
16:      return ;
17:   finsi
18:   sum_H ← sum_H + sum_I' ;
19:   sum_I ← sum_I + d ;
20: fin pour
21: si D(H ∪ {I}) > D(H) alors
22:   Notifier I ;
23:   H ← H ∪ I ;
24:   Mise à jour des sommes des distances ;
25: finsi
```

Lors la même passe, nous calculons la somme des distances du nouvel item publié sum_{I_n} avec les items de l'historique :

$$sum_{I_n} = \sum_{I \in H} dist(I_n, I)$$

Après avoir vérifié la nouveauté de I_n, nous testons s'il favorise la diversité de l'ensemble d'items de l'historique. La distance moyenne entre les items de H $D(H)$ et celle des items de l'historique une fois I_n ajouté $D(H \cup I_n)$ sont comparées. $D(H)$ et $D(H \cup I_n)$ sont calculées par les formules suivantes :

$$D(H) = \frac{2}{|H| * (|H| - 1)} \times sum_H$$

$$D(H \cup I_n) = \frac{2}{(|H|+1) * |H|} \times (sum_H + sum_{I_n})$$

Si I_n favorise la diversité $D(H \cup \{I_n\}) > D(H)$, I_n sera notifié à l'utilisateur et ajouté à H. Nous devrons alors mettre à jour sum_I, la somme des distances de chaque item $I \in H$, en l'incrémentant de sa distance à I_n ($dist(I_n, I)$).

La complexité du calcul du $D(H)$ et $D(H \cup I_n)$ est de l'ordre de O($|H|$), tandis que la complexité du calcul de la distance moyenne entre les items de H par la méthode classique est de l'ordre de O($|H|^2$). Cette optimisation de calcul permet un gain important en temps de traitement.

Nous profitons pour cette optimisation du fait que les items sont ajoutés à l'historique suivant leur ordre inverse d'arrivée. Comme le montrent les instructions des lignes 3-4 de l'Algorithme 6, les comparaisons avec les items de l'historique sont arrêtées dès qu'il y a un item qui n'est plus dans la fenêtre, permettant d'ignorer le plus tôt possible tous les items inutiles au calcul.

5.5 Expériences : Evaluation du filtrage

Dans cette section, nous allons étudier l'impact de différents paramètres comme le seuil de nouveauté et la taille de la fenêtre sur le taux de filtrage par nouveauté et diversité pour un jeu réel d'items. Ensuite, nous mesurons les performances du filtrage : besoin en espace mémoire et le temps de traitement des items. Enfin, nous comparons notre approche pour le filtrage sur une fenêtre glissante avec un filtrage périodique basé sur un algorithme *top-k*.

5.5.1 Implantation

Nous avons implanté les historiques des souscriptions en utilisant la plateforme standard de Java (*version Java* « 1.6.0_20 »). Et les expériences ont été exécutées avec un quadri-cœur processeur 3, 60 GHz et un espace mémoire de 16 Go pour la JVM.

Pour chaque souscription, une liste chaînée de tableaux est associée pour stocker les liens (pointeurs) vers les items de son historique. Un tableau de taille égale au nombre de souscriptions indexées dans le système est utilisé pour sauvegarder un pointeur vers la tête de la liste de chaque souscription (voir la figure 5.5). Les éléments des listes chaînées contiennent un autre tableau de type *double* (8 octets) de la même taille du tableau de pointeurs vers les items. Une entrée est associée à chaque item pour stocker la somme de ses distances avec les items plus récents dans l'historique de la souscription. Les items sont des instances d'une classe composée de : ($item_{id}, termes_{tab}, timestamp$), où $item_{id}$ est l'identificateur de l'item codé sur un entier de 4 octets (int), $termes_{tab}$ est un tableau

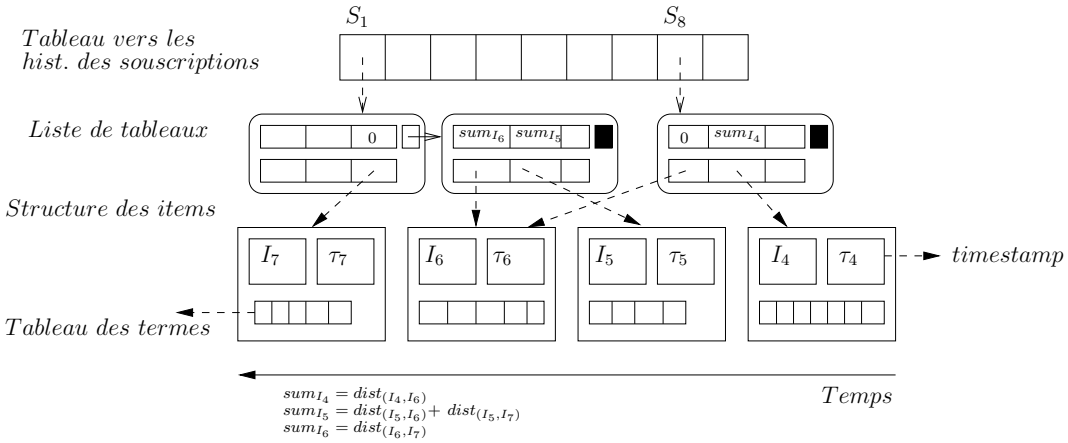

Figure 5.5 – Implantation des historiques des souscriptions

d'entiers (*int*) pour stocker les identificateurs des termes de l'item et *timestamp* est un entier de type *long* (codé sur 8 octets) pour sauvegarder l'instant de publication de l'item.

5.5.2 Description des jeux de données

Pour ces expériences, nous utilisons un sous-ensemble d'items de notre jeu de données réel de flux RSS acquis pendant une période de huit mois de mars à octobre 2010 (voir chapitre 3). L'ensemble d'items acquis pendant la première semaine d'octobre (258 480 items) est utilisé avec des dates de publication qui correspondent à leur date d'acquisition (*timestamp*).

La méthode Alias [Walker, 1977] utilisée dans la section 4.5.2 est également utilisée pour la génération de 10 millions de souscriptions en utilisant le vocabulaire de 1,5 millions de termes distincts extraits des items. La distribution des tailles des souscriptions respecte la distribution des tailles des requêtes sur le Web [Vouzoukidou, 2010, Beitzel et al., 2004]. Elle est caractérisée par une taille maximale de 12 et une taille moyenne égale à $2,2$. À noter que seulement 5,28 millions de souscriptions sont satisfaites au moins une fois pendant la semaine étudiée avec une hypothèse de *broad match* (voir Définition 1), dont 4,16 millions sont satisfaites au moins une fois le dernier jour.

5.5.3 Taux de filtrage

Plusieurs paramètres influent sur le taux de filtrage des items. Nous étudierons dans cette section l'impact du seuil de nouveauté et la diversité. La taille de la fenêtre impacte également le filtrage. Les résultats présentés dans cette section sont les valeurs moyennes de taux de filtrage par souscription pendant le dernier jour de la semaine (4,16 millions de souscriptions).

Impact du seuil de nouveauté

La figure 5.6 montre le taux de filtrage par nouveauté en faisant varier le seuil de nouveauté pour une taille de fenêtre fixée à 24 heures. Nous remarquons que le filtrage augmente avec le seuil de nouveauté. Pour un seuil de nouveauté égal à 50%, 40% des items qui satisfont la souscription sont filtrés. Cela montre que la moitié de l'information publiée dans un nombre important d'items est redondante. Nous rappelons que pour calculer la nouveauté d'un item nous prenons en considération l'importance (poids) des nouveaux termes qui y appartiennent (Définition 7 de nouveauté item-item). En moyenne, seulement 20% des items qui satisfont une souscription ne sont constitués que de nouvelle information (nouveauté égale à 100%).

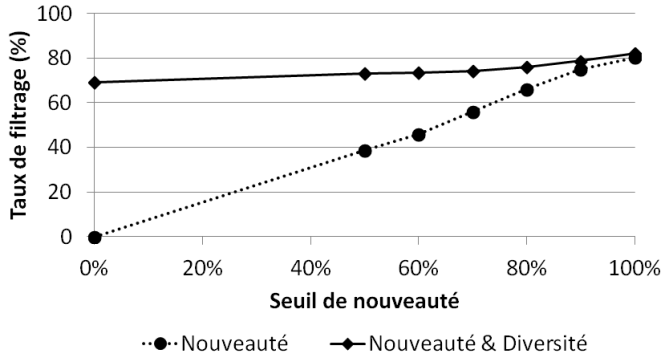

Figure 5.6 – Taux de filtrage en fonction du seuil de nouveauté

Impact de la diversité

Quel que soit le seuil de nouveauté, le filtrage par diversité réduit le nombre d'items à notifier (voir figure 5.6). L'impact du filtrage par diversité est plus important que celui par nouveauté, mais cela ne néglige pas le filtrage par nouveauté. Nous pouvons remarquer que le filtrage par diversité seul (*c.-à-d.*, quand le seuil de nouveauté est égal à 0%) est inférieur au filtrage par nouveauté quelque soit le seuil de nouveauté ($\sim$ 69%). En plus, pour tous les seuils de nouveauté le filtrage avec la diversité est supérieur au filtrage par nouveauté seul. Donc, il y a des items filtrés par nouveauté et qui ne sont pas filtrés par la diversité et inversement. Cela justifie la distinction entre les deux types de filtrage et les différentes définitions de la diversité (section 2.3.1). En plus, le filtrage par nouveauté et diversité devient plus important que celui par diversité seule pour un seuil de nouveauté égal à 80%.

Par la suite, nous fixerons le seuil de nouveauté par défaut à 50% en prenant en considération la diversité pour le filtrage.

Impact de la taille de la fenêtre

La figure 5.7 montre le taux de filtrage pour un seuil de nouveauté fixé à 50% pour différentes tailles de la fenêtre. En augmentant la taille de la fenêtre, le taux de filtrage devient plus important, car le nombre d'items à prendre en considération pour calculer la nouveauté et la diversité d'un nouvel item publié est plus important. Nous allons voir plus tard qu'avec le temps il y aura une stabilité des tailles des historiques des souscriptions (le nombre d'items dans la fenêtre). Ainsi, le nombre d'items dans de larges fenêtres pourra être inférieur ou égal à celui dans de petites fenêtres (voir figure 5.13). En fait, les items dans les larges fenêtres sont très sélectifs et restent plus de temps d'où ils provoquent le filtrage d'un nombre plus grand d'items.

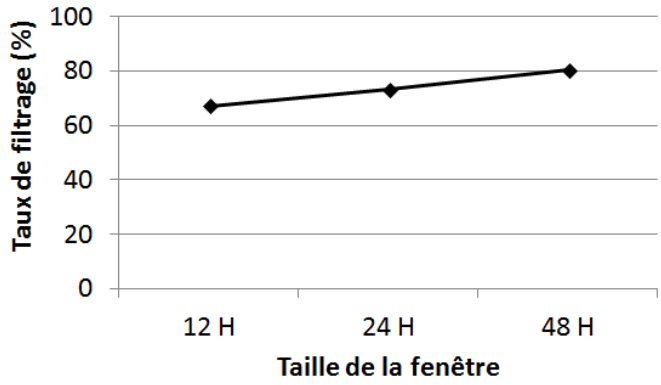

Figure 5.7 – Taux de filtrage en fonction de la taille de la fenêtre

Nous avons aussi remarqué qu'au cours du temps il y a une variation du taux de filtrage par taille de fenêtre. Nous l'avons aussi mesurée en fonction du temps pour différentes tailles de la fenêtre et un seuil de nouveauté fixé à 50% (voir figure 5.8). Afin d'étudier la variation du taux de filtrage en fonction du temps, nous l'avons mesuré pour des intervalles de temps de six heures sur toute la semaine, pour les souscriptions qui sont satisfaites au moins une fois pendant les six premières heures de la semaine (3,35 millions de souscriptions). Nous remarquons une certaine stabilité avec un taux compris entre 60 et 80%, et un comportement similaire pour les différentes tailles de la fenêtre. Le fait que les items notifiés avec de larges fenêtres sont sélectifs garantit que le taux de filtrage est toujours plus important avec les larges fenêtres.

Impact de la taille des souscriptions

Le tableau 5.3 présente la distribution du nombre de souscriptions par taille qui compose notre jeu de souscriptions. Cette distribution respecte celle des requêtes sur le Web observée dans [Vouzoukidou, 2010, Beitzel et al., 2004]. Nous remarquons ainsi que la majorité des souscriptions ont une taille inférieure à 4.

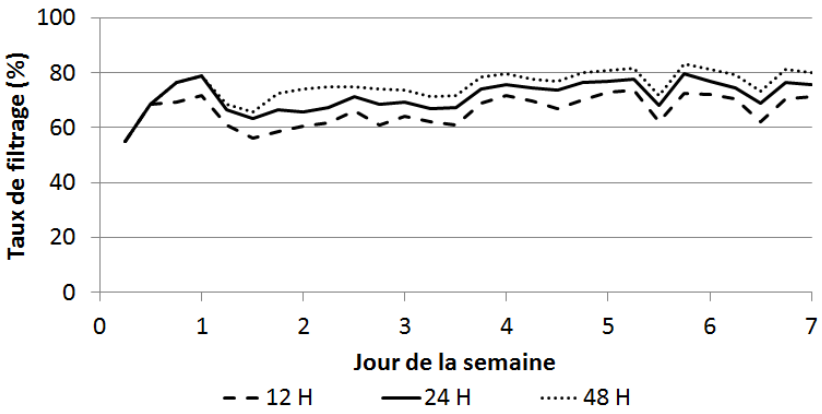

Figure 5.8 – Variation du taux de filtrage en fonction du temps

$\|s\|$	Nombre de souscriptions
1	2 030 375
2	1 804 265
3	293 666
>3	28 776

Table 5.3 – Distribution des tailles des souscriptions

Ce qui différencie le comportement des souscriptions de différentes tailles est leur taux de satisfaction. Le tableau 5.4 montre le pourcentage de souscriptions par taille et nombre de notifications (*c.-à-d.*, satisfactions ou bien débit de réception par l'utilisateur après le *matching*), ainsi que le nombre de notifications moyen par taille. La majorité des souscriptions de taille 1 sont souvent notifiées (> 50 items/jour). Les souscriptions de taille égale à 2 sont moyennement notifiées. Les larges souscriptions ont des taux de notifications inférieurs à 5 items/jour.

$\|s\|$	1-5	6-10	11-50	51-100	101-500	501-1000	>1000	# de notif. moyen
1	6,32%	3,63%	15,06%	10,80%	37,07%	13,70%	13,42%	505,31
2	46,39%	15,98%	28,08%	5,76%	3,65%	0,07%	0,07%	21,94
3	77,93%	12,03%	9,35%	0,58%	0,11%	0,0003%	0,003%	4,98
>3	92,25%	5,47%	2,24%	0,04%	0%	0%	0%	2,45

Table 5.4 – Pourcentage de souscriptions en fonction du nombre de notifications et de la taille des souscriptions pendant les dernières 24 heures

La figure 5.9 montre le taux de filtrage pour différentes tailles de souscriptions pour une taille de fenêtre égale à 24 heures et un seuil de nouveauté fixé à 50%. Le filtrage est plus important pour les petites souscriptions ($|s| = 1$). En effet, le nombre d'items qui satisfont ces souscriptions étant plus grand (voir tableau 5.4), le filtrage est plus important puisque

la probabilité que deux items qui satisfont une souscription soient similaires augmente. De plus, les items qui satisfont les petites souscriptions sont en moyenne plus courts ($\sim$ 73,82 termes) par rapport aux items qui satisfont les larges souscriptions ($\sim$ 520,41 termes). Ces items ne contiennent pas de nouvelle information et ne favorisent donc pas la diversité. Par conséquent, ils sont plus filtrés. Il est à noter que comme le nombre de souscriptions dont la taille est égale à 1 est le plus grand, les résultats moyens sur l'ensemble de souscriptions sont plus influencés par leur taux de filtrage.

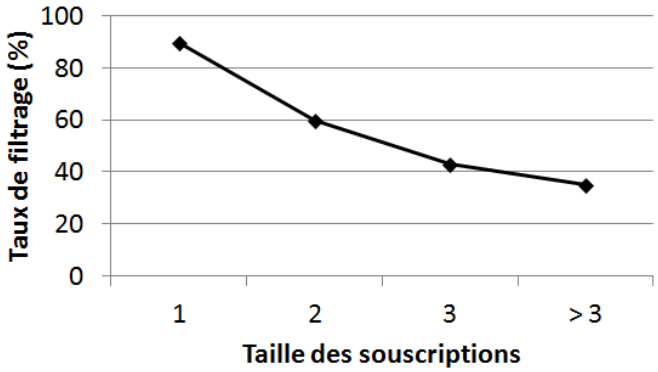

Figure 5.9 – Taux de filtrage en fonction de la taille des souscriptions

La figure 5.10 montre le taux de filtrage en fonction du nombre de notifications pour différentes tailles de fenêtre avec un seuil de nouveauté fixé à 50%. Nous rappelons que le nombre de notifications par souscription est égal au nombre d'items satisfaisant la souscription pendant le dernier jour de la semaine. Le filtrage est plus important pour les souscriptions dont le nombre de notifications est élevé. Nous avons déjà observé que le taux de filtrage augmente avec la taille de la fenêtre, et d'après ces résultats nous remarquons que l'augmentation est plus importante quand le nombre de notifications est petit. Nous pouvons conclure que la taille de la fenêtre n'a pas d'impact sur le taux de filtrage lorsque la souscription est très notifiée ($>$ 500 items/jour).

Taille des historiques des souscriptions

Les tailles des historiques des souscriptions influent sur les besoins en espace mémoire et le temps de traitement, pour cela nous étudions la variation des tailles des historiques, en nombre d'items, pour différents paramètres : la taille des souscriptions, le seuil de nouveauté et la taille de la fenêtre.

Le nombre de notifications en entrée diminue avec la taille des souscriptions (tableau 5.4) par conséquent la taille des historiques diminue avec la taille des souscriptions. La figure 5.11 présente la taille moyenne des historiques en fonction de la taille des souscriptions pour un seuil de nouveauté égal à 50% et une fenêtre de 24 heures. Nous remarquons que les historiques des souscriptions de taille 1 sont à peu près dix fois plus larges que celles

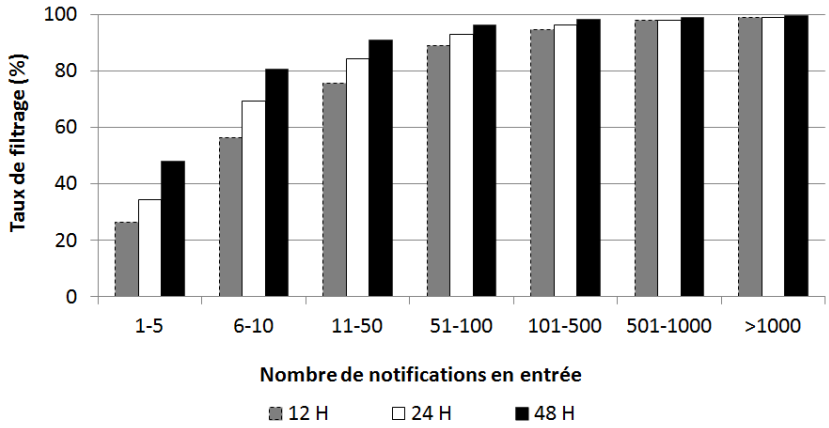

Figure 5.10 – Taux de filtrage en fonction du nombre de notifications

des larges souscriptions. Bien que les courtes souscriptions sont $\sim$ 250 fois plus notifiées que les larges souscriptions, cela montre que le filtrage n'est pas linéaire avec le nombre de notifications.

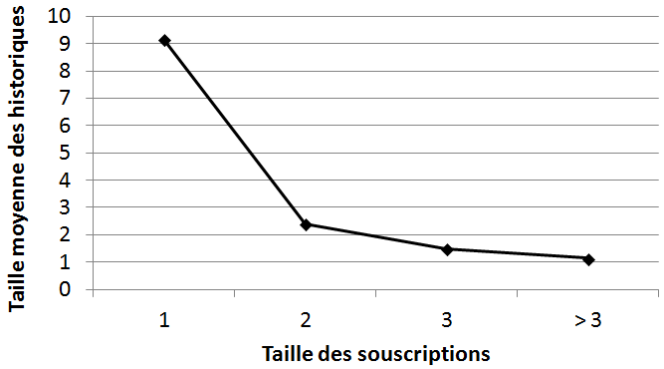

Figure 5.11 – Taille des historiques en fonction de la taille des souscriptions

La figure 5.12 montre le nombre moyen d'items dans les historiques des souscriptions en fonction du seuil de nouveauté et pour une fenêtre égale à 24 heures. La taille des historiques des souscriptions diminue avec le seuil de nouveauté, car le filtrage est plus important.

Nous nous intéressons également à étudier la taille des historiques en fonction de la taille de la fenêtre, pour cela nous avons mesuré la variation de la taille des historiques en fonction du temps entre des intervalles de temps de six heures sur toute la semaine. La figure 5.13 montre cette variation pour différentes tailles de la fenêtre et un seuil de nouveauté fixé à 50%, les valeurs présentées sont les valeurs moyennes des tailles des historiques des souscriptions qui sont satisfaites au moins une fois pendant les six premières heures de la semaine (3,35 millions de souscriptions). Il faut noter aussi que

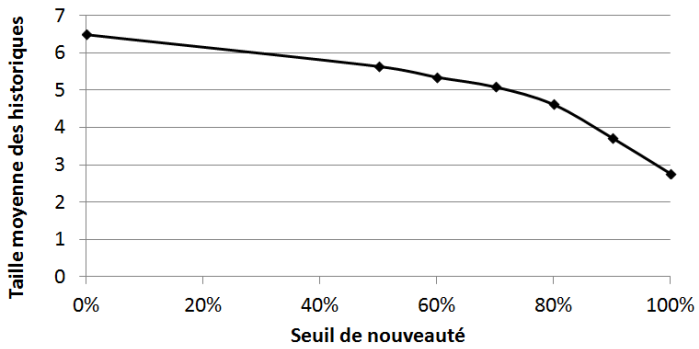

Figure 5.12 – Taille des historiques en fonction du seuil de nouveauté

les tailles des historiques à un instant τ sont égales au nombre d'items dans la fenêtre de taille considérée p (items publiés après $\tau - p$).

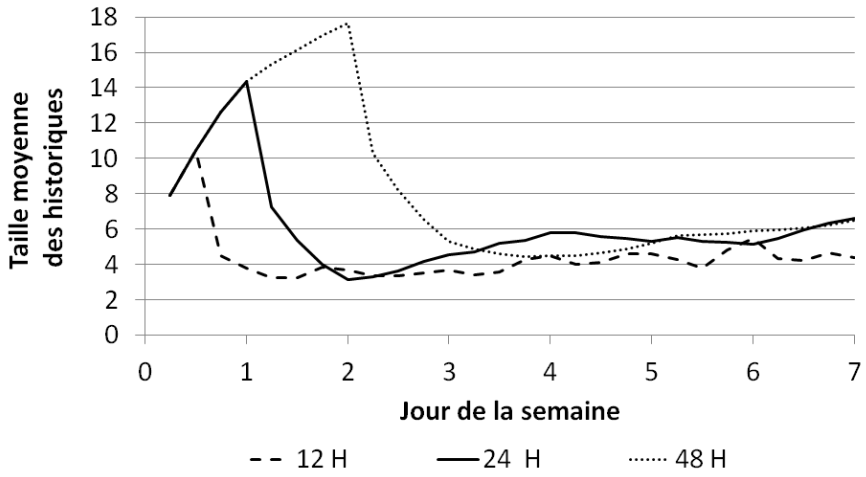

Figure 5.13 – Variation de la taille des historiques en fonction du temps

Lors de la phase d'initialisation, nous observons que la taille des historiques est plus grande pour les grandes fenêtres. Pour chaque taille de fenêtre la taille moyenne des historiques atteint un pic à la fin de la première fenêtre correspondant à la fenêtre d'un historique. La raison est que l'historique ne couvrant pas une période entière de taille d'historique et il ne va pas filtrer autant que sur une fenêtre correspondant à une période et donc contenant plus d'items. Au cours du temps, nous observons que la taille des historiques varie suivant la taille de la fenêtre uniquement pour l'initialisation. Par contre par la suite la taille est stable en fonction du temps et ne dépend plus de la taille de la fenêtre.

Nous avons aussi mesuré cette variation pour un seuil de nouveauté égal à 80% (voir figure 5.14). Les résultats confirment notre conclusion, avec des amplitudes différentes

pour la phase d'initialisation. Ce comportement est dû donc au filtrage par diversité.

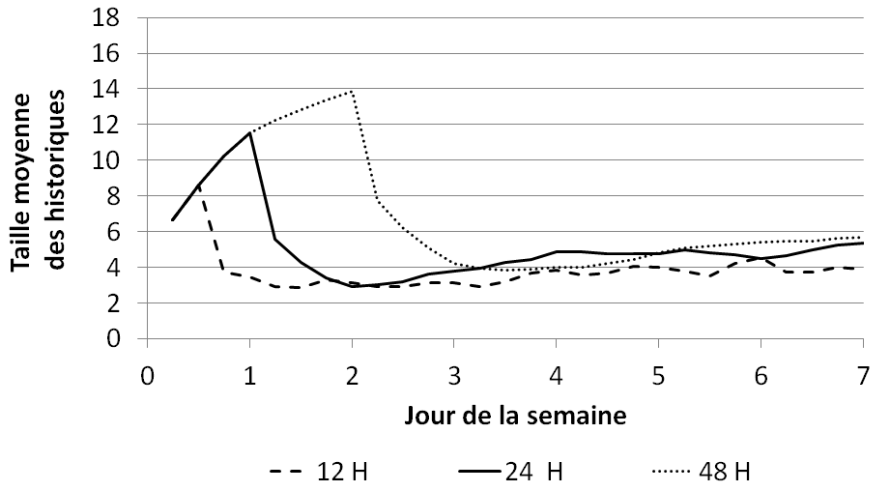

Figure 5.14 – Variation de la taille des historiques en fonction du temps pour un seuil de nouveauté de 80%

La figure 5.15 présente la variation des tailles des historiques en fonction du nombre de notifications pour différentes tailles de fenêtre et un seuil de nouveauté égal à 50%. Quand le nombre de notifications est modéré ($\leq$ 500 items/jour), la taille des historiques augmente avec la taille de la fenêtre. Mais avec un nombre élevé de notifications où le taux de filtrage est plus important (voir figure 5.10) la taille des historiques diminue pour de larges fenêtres. D'après les résultats des figures 5.10 et 5.15, nous pouvons garantir que notre système permet le passage à l'échelle pour des débits d'arrivée d'items élevés.

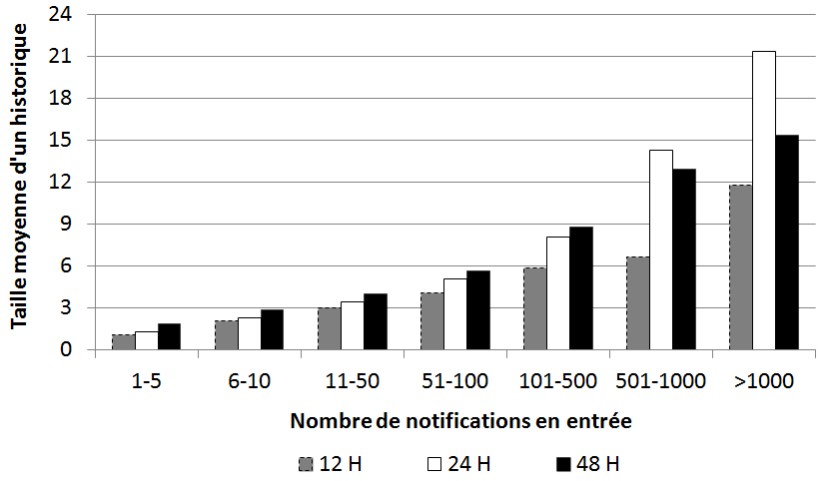

Figure 5.15 – Taille des historiques en fonction du nombre de notifications

Afin de pouvoir intérpreter les performances du notre filtrage, nous avons mesuré la taille moyenne des historiques à la fin de la semaine pour les souscriptions qui sont satisfaites au moins une fois pendant les dernières 24 heures de la semaine. La figure 5.16 présente la taille des historique pour plusieurs tailles de la fenêtre pour un seuil de nouveauté de 50%. Nous remarquons que la taille des historiques est indépendante de la taille de la fenêtre.

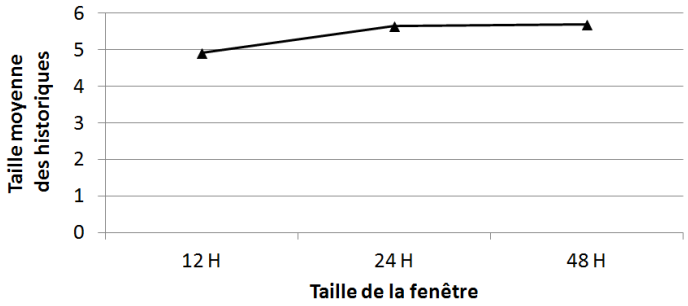

Figure 5.16 – Taille des historiques en fonction de la taille de la fenêtre

5.5.4 Evaluation des performances

Nous nous intéressons dans cette section à étudier les performances en espace mémoire et en temps du traitement ainsi que le passage à l'échelle. Nous comparons les performances de l'implantation classique du système de filtrage qui consiste à recalculer la densité (distance moyenne) de l'ensemble d'items des historiques à l'arrivée d'un item aux performances du filtrage avec notre optimisation qui consiste à ne pas recalculer la densité en cumulant la somme des distances de l'item avec les autres items plus récents dans l'historique pour calculer la distance moyenne (section 5.4.5).

Avant d'étudier les performances, nous voulons choisir la bonne valeur pour la capacité des tableaux des listes chaînées pour les historiques des souscriptions qui n'influe pas sur les résultats. Le tableau 5.5 montre l'espace mémoire utilisé et le temps de traitement pour différentes valeurs de la capacité des tableaux pour un seuil de nouveauté de 50% et une fenêtre égale à 24 heures. Nous remarquons qu'une valeur égale à 20 est un bon compromis entre l'espace mémoire perdu dans les cases vides des tableaux et le temps de traitement qui comprend notamment le temps nécessaire pour l'allocation de nouveaux tableaux durant le traitement. Nous avons vu dans la section précédente que la taille des historiques dépend du nombre de notifications en entrée lui-même lié à la taille de la souscription. Nous avons également proposé afin de réduire l'espace mémoire perdu par les cases vides des tableaux de varier la capacité des tableaux en fonction de la taille de la souscription. Pour une souscription de taille $|s|$, la capacité des tableaux de son historique est fixée à $20/|s|$. D'après les résultats du tableau 5.5, nous gagnons par rapport à la capacité de 20 choisie comme compromis en espace mémoire pour une faible dégradation du temps de traitement.

Capacité	Mémoire (Mo)	Temps (ms)
10	1 655,61	27,32
15	1 735,86	26,49
20	1 799,27	25,66
25	1 955,20	25,67
$20/\|s\|$	1 750,65	25,73

Table 5.5 – Mémoire et temps pour différentes capacités des tableaux

Espace mémoire

La figure 5.17 montre l'espace mémoire utilisé par les historiques pour différentes valeurs de seuil de nouveauté pour une taille de fenêtre égale à 24 heures. Nous observons que l'espace consommé par les deux implantations diminue avec le seuil de nouveauté, car il y a plus de filtrage d'où des historiques plus courts. L'espace mémoire utilisé avec notre optimisation du calcul du densité est plus grand, car nous avons besoin de sauvegarder pour chaque item de l'historique la somme de ses distances avec les items plus récents. L'espace mémoire utilisé est inversement proportionnel au filtrage (voir figure 5.6). Nous remarquons que le taux de filtrage passe de 69,10% pour un seuil de nouveauté égal à 0% à 82,05% pour 100% de nouveauté soit une augmentation de 18,74%, alors que l'espace mémoire diminue 1 848,22 Mo pour 0% de nouveauté à 1 567,72 Mo pour 100% soit une baisse de 17,89%.

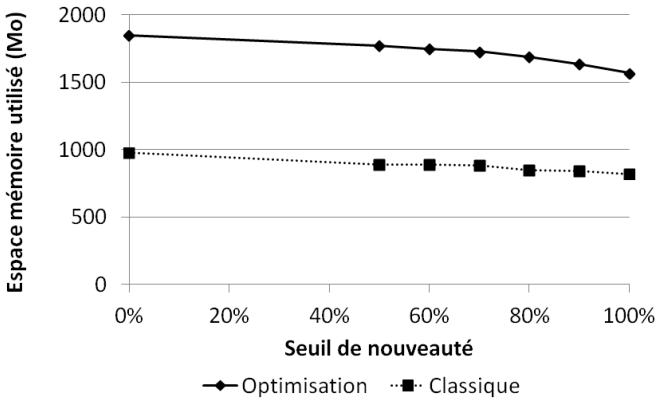

Figure 5.17 – Espace mémoire utilisé en fonction du seuil de nouveauté

La figure 5.18 présente la variation de l'espace mémoire en fonction de la taille de la fenêtre, nous observons que la taille de la fenêtre n'impacte pas l'espace mémoire utilisé. En effet, bien que le taux de filtrage augmente avec la taille de la fenêtre (voir figure 5.7), nous avons vu dans la figure 5.13 que les tailles des historiques sont constantes et donc il en est de même pour la consommation en espace mémoire.

La figure 5.19 montre l'espace mémoire en faisant varier le nombre de souscriptions pour

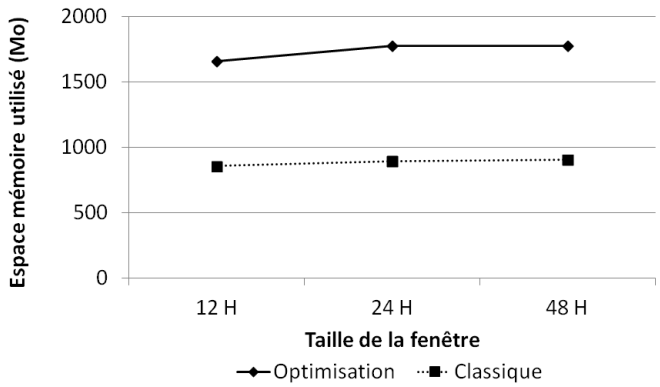

Figure 5.18 – Espace mémoire utilisé en fonction de la taille de la fenêtre

un seuil de nouveauté fixé à 50% et une fenêtre égale à 24 heures. La génération de nos souscriptions respecte la distribution des fréquences des termes dans les items d'où un taux de notifications (satisfactions) moyen par souscription constant et donc la taille moyenne des historiques indépendante du nombre de souscriptions. Nous remarquons que l'espace mémoire augmente linéairement en fonction du nombre de souscriptions indexées.

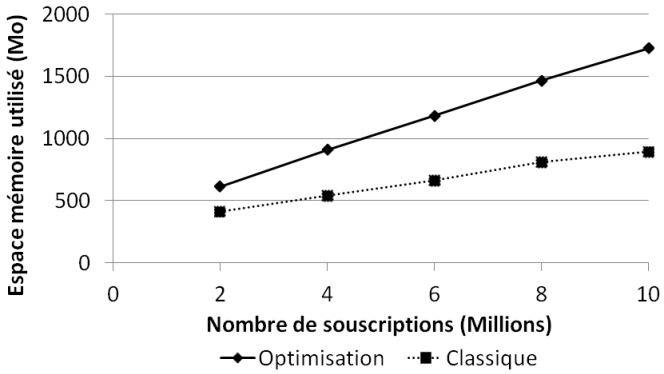

Figure 5.19 – Espace mémoire utilisé en fonction du nombre de souscriptions

Temps de traitement

Le temps de traitement diminue lorsque le seuil de nouveauté croît puisque les historiques sont plus courts (figure 5.20). Il est plus important avec l'implantation classique du complexité $O(|H|^2_{moy})$ où $|H|_{moy}$ est la taille moyenne des historiques. Par contre pour notre optimisation, le calcul consiste à calculer la somme des sommes des distances de chaque item de l'historique ($O(|H|_{moy})$). Cette différence diminue avec le seuil de nouveauté car les historiques deviennent plus courts. En effet, un seuil de nouveauté élevé permet un rejet plus tôt de l'item d'où moins de parcours des historiques et de calcul de diversité

pour les deux implantations.

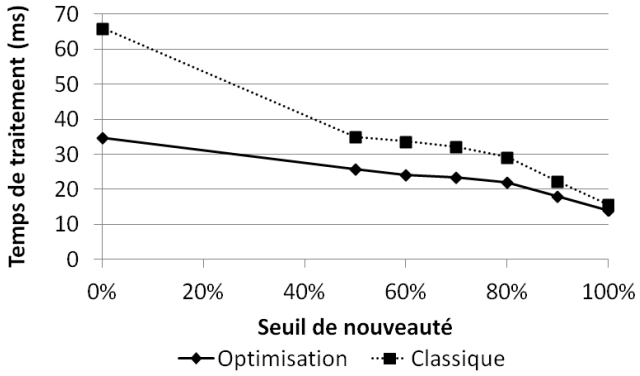

Figure 5.20 – Temps de traitement en fonction du seuil de nouveauté

Le temps de traitement est principalement affecté par la taille des historiques à parcourir et à traiter. Comme nous avons vu dans la section précédente, la taille moyenne des historiques (voir figure 5.16) est stable quelque soit la taille de la fenêtre. Nous observons dans la figure 5.21 qui présente la variation du temps de traitement pour différentes tailles de la fenêtre, que le temps de traitement pour notre optimisation est indépendant de la taille de la fenêtre.

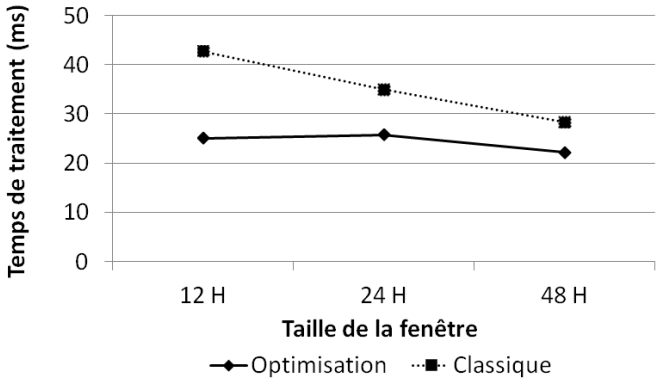

Figure 5.21 – Temps de traitement en fonction de la taille de la fenêtre

Regardant le temps de traitement pour le calcul classique, il diminue lorsque la taille de la fenêtre est plus grande. En effet, nous avons également implanté le calcul classique d'une manière optimisée. Au lieu de recalculer la distance moyenne entre les items de l'historique à l'arrivée de chaque item, nous cumulons la distance du nouvel item avec les items de l'historique en même temps que nous calculons sa nouveauté, si aucun item ne quitte l'historique (contrainte du temps sur la fenêtre glissante) nous cumulons cette somme avec la somme des distances entre les items que nous avons déjà stocké pour avoir la nouvelle densité de l'historique, si au moins un item quitte l'historique il faut

recalculer la distance moyenne. Nous avons remarqué que nous gagnons beaucoup en temps de traitement avec cette optimisation, en moyenne pour différentes tailles de la fenêtre et un seuil de nouveauté égal à 50%, seulement 1,66% des calculs des densités sont faits. Bien sûr ce pourcentage augmente avec la taille de la fenêtre, car pour les petites fenêtres le nombre d'items qui quittent la fenêtre est plus grand. Comparant le nombre de calculs pour les différentes tailles de la fenêtre, en passant d'une fenêtre de 12 heures à une de 24 heures le nombre de calculs diminue de 33% puis en passant à une de 48 heures il diminue de 36%. Ayant une stabilité dans la taille des historiques en fonction de la taille de la fenêtre et un nombre de calculs qui diminue avec la taille de la fenêtre le temps de traitement diminue avec la taille de la fenêtre (figure 5.21).

Enfin, le temps de traitement augmente linéairement avec le nombre de souscriptions (figure 5.22) pour les deux implantations. Avec notre optimisation pour les calculs de la nouveauté et des distances entre les items, nous nous attendons à une augmentation sous-linéaire car cette partie du traitement sera commune entre les souscriptions ce qui n'est pas le cas. Donc, nous pouvons conclure que le temps de calcul de la nouveauté de l'item et de sa distance avec les items des historiques est négligeable par rapport au temps de parcours des historiques. Nous obtenons un gain moyen de 28,30% en temps de traitement avec l'optimisation du calcul de la densité des items des historiques.

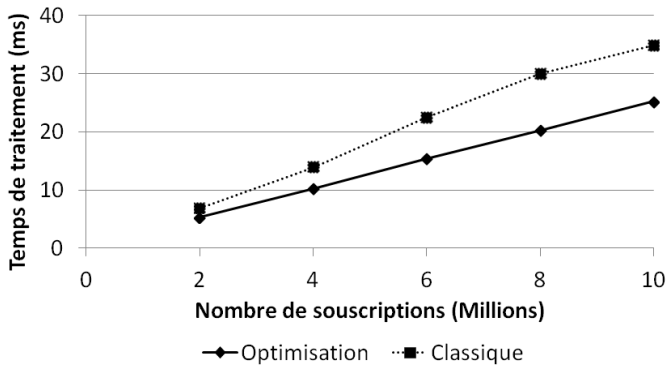

Figure 5.22 – Temps de traitement en fonction du nombre de souscriptions

5.5.5 Qualité du filtrage

Fenêtre glissante vs. filtrage périodique

Afin d'étudier la qualité de notre filtrage, nous avons comparé l'ensemble d'items notifiés à l'utilisateur après le filtrage avec celui choisi par une approche *top-k*. Pour cela, nous avons implanté un algorithme *Greedy* (permettant le passage à l'échelle du calcul de *top-k*) pour calculer les k items les plus distants parmi les items qui satisfont la souscription. L'ensemble résultat est initialisé par les deux items les plus distants parmi les items satisfaisant la souscription. La valeur de k est fixée par souscription, elle est égale au nombre

d'items notifiés par notre approche basée sur une fenêtre glissante, cela nous permet une bonne comparaison entre les deux ensembles résultats. À noter que cet algorithme donne un *top-k* approché. Par ailleurs, nous avons choisi de comparer l'ensemble de k items notifiés par notre approche par rapport aux ensembles fournis par un *top-(k*1,2)* et un *top-(k*1,5)*.

Nous mesurons la qualité de notre approche en calculant l'intersection entre notre résultat et celui de *top-k Greedy*. Comme l'algorithme *Greedy* ne prend pas en considération la nouveauté, pour cela nous nous intéressons à les comparer pour le filtrage par diversité où le seuil de nouveauté est égal à 0%.

	12 H	24 H	48 H
top-k	60,17%	62,84%	68,20%
top-(k*1,2)	66,90%	69,96%	75,41%
top-(k*1,5)	72,08%	73,33%	77,98%

Table 5.6 – Ratio de l'intersection pour le filtrage par diversité

Le tableau 5.6 présente l'intersection pour différentes tailles de la fenêtre quand le seuil de nouveauté est fixé à 0%. Plus la fenêtre est large, plus l'intersection est importante. Nous trouvons qu'au moins 60% des items sélectionnés sont aussi sélectionnés par un algorithme *top-k Greedy*. En plus, 78% des items de notre approche sont parmi les *top-(k*1,5)* items.

Nous avons également étudié l'impact du seuil de nouveauté sur la qualité de notre notification (tableau 5.7). Nous trouvons que l'intersection donc la qualité diminue lorsque le seuil de nouveauté augmente, car il y a des items qui sont filtrés par la nouveauté alors qu'ils ne le sont pas par la diversité.

	0%	50%	80%	100%
top-k	63,04%	60,18%	52,65%	43,29%
top-(k*1,2)	70,22%	67,60%	60,88%	50,82%
top-(k*1,5)	73,70%	71,00%	64,34%	54,03%

Table 5.7 – Ratio de l'intersection pour différents seuils de nouveauté

Comme le but de notre filtrage est d'envoyer à l'utilisateur des items dont le contenu est varié (*c.-à-d.*, les plus distants). Nous avons choisi une autre mesure de qualité basée sur la distance moyenne entre les items de deux ensembles. Nous avons mesuré le pourcentage de souscriptions pour lesquelles la distance moyenne entre les items fournis par notre approche est supérieure ou égale à celle des items choisis par l'algorithme *top-k*. Pour un filtrage par diversité (nouveauté égale à 0%) sur une fenêtre de 48 heures, pour 33,09% des souscriptions le résultat fourni par notre approche possède une distance moyenne supérieure ou égale à l'approche *top-k*. De plus, 79,60% des souscriptions sont notifiées par un ensemble d'items ayant une distance moyenne entre eux à 0,1 près de celle de l'ensemble fourni par l'approche *top-k*.

Nous avons vu dans la section 5.5.3 qu'il y a des items filtrés par nouveauté et qui ne le sont pas par la diversité. Et comme dans l'approche *top-k* basée sur l'algorithme *Greedy*, le filtrage est seulement fait par diversité nous avons étudié le taux de redondances dans le résultat choisi. Nous avons calculé le pourcentage des items notifiés à la souscription qui pourraient être supprimés par notre approche. Notre filtrage par nouveauté est basé sur l'ordre d'arrivée des items et une mesure asymétrique de nouveauté. Pour avoir les mêmes conditions de calcul de la nouveauté d'un item notifié par l'approche *top-k*, nous avons classé les items choisis par leur ordre d'arrivée et jugé qu'un item pourraît être supprimé par nouveauté s'il existe au moins un item plus ancien qui ne prouve pas sa nouveauté. Nous avons fait des expériences pour différents seuils de nouveauté et taille de fenêtre égale à 24 heures. Nous avons trouvé qu'en moyenne par souscription seulement 2,86% des items notifiés ne dépassent pas le seuil de 50% de nouveauté, tandis que 5,85% ne sont pas nouveaux à 80%. 17,75% des items notifiés contiennent de l'information redondante (*c.-à-d.*, ne sont pas nouveaux à 100%).

D'après ces résultats, nous trouvons que le filtrage par diversité n'est pas suffisant pour envoyer un résultat de bonne qualité à l'utilisateur. De plus, nous pouvons comprendre pourquoi les deux ensembles des résultats de notre approche et celui de l'approche *top-k* ne sont pas proches en prenant en considération la nouveauté (voir tableau 5.7). Cela est dû au fait qu'il y a des items favorisant la diversité, tout en n'étant pas nouveaux qui sont filtrés par notre approche et non par celle de *top-k*.

Résumé

Nous concluons de notre ensemble d'expériences que :

i) le taux de filtrage augmente avec le seuil de nouveauté. La diversité est un filtrage supplémentaire. Pour un seuil de nouveauté de 50% et en prenant en considération la diversité nous filtrons $\sim$ 70% des items. De plus, ce filtrage devient plus important avec de larges fenêtres ;

ii) la taille des historiques diminue lorsque les seuils de nouveauté augmente mais elle est indépendante de la taille de la fenêtre ;

iii) le nombre d'items qui satisfont la souscription influe sur le taux de filtrage et par conséquent sur la taille de son historique. L'augmentation du taux de filtrage avec la taille de la fenêtre est plus significative pour les souscriptions peu satisfaites. La taille des historiques augmente avec la taille de la fenêtre pour les souscriptions peu satisfaites tandis que celle des souscriptions très satisfaites elle est presque indépendante ;

iv) l'espace mémoire utilisé par les historiques et le temps de traitement diminuent avec le seuil de nouveauté et sont linéaires en fonction du nombre de souscriptions indexées, et indépendants de la taille de la fenêtre ;

v) parmi les items notifiés après le filtrage par diversité, en moyenne pour différentes tailles de la fenêtre 64% de ces items sont aussi notifiés par une approche *top-k Greedy*. De plus, pour 33% des souscriptions notre méthode donne un ensemble d'items qui est

de la même qualité ou meilleure (plus distant) que le *top-k* approché par l'algorithme *Greedy*.

5.6 Conclusion

Dans ce chapitre, nous avons défini un algorithme de filtrage des items par nouveauté et diversité de l'information par rapport à un historique produit pour chaque souscription. La nouveauté est filtrée item par item alors que la diversité se mesure sur l'ensemble de l'historique. Nous avons choisi une pondération des termes s'appuyant sur une mesure de TDV. Afin de réduire l'espace pris par l'ensemble des items des historiques, une unique fenêtre d'items est préservée en mémoire et chaque souscription maintient une liste de pointeurs vers les items de cette fenêtre constituant son historique. Nous avons étudié le taux de filtrage par nouveauté et diversité pour un jeu réel d'items pour différents paramètres. Le taux de filtrage augmente avec le seuil de nouveauté et la taille de la fenêtre et devient plus important avec la diversité. Il est dépendant du nombre d'items reçus par l'utilisateur pour sa souscription, il est plus important pour les souscriptions très satisfaites. Les performances du filtrage ont été aussi étudiées, nous avons trouvé que l'espace mémoire utilisé par les historiques et le temps de traitement diminuent avec le seuil de nouveauté. Ils sont linéaires en fonction du nombre de souscriptions indexées, tandis qu'ils sont indépendants de la taille de la fenêtre. En comparant les performances avec celles d'une approche classique pour calculer la distance moyenne entre les items de l'historique, malgré l'augmentation de la consommation en espace mémoire, nous gagnons en moyenne 28,30% en temps de traitement. Nous avons aussi comparé notre approche pour le filtrage sur une fenêtre glissante avec un algorithme *top-k* périodique *Greedy*, nous avons trouvé qu'environ 70% des items notifiés par notre approche pour la diversité sont également notifiés par l'approche *top-k*. De plus, 33% des souscriptions sont notifiées par un ensemble d'items dont la distance moyenne est supérieure ou égale à celle de l'ensemble choisi par *top-k*.

Chapitre 6

Conclusion

Dans ce chapitre, nous présentons les principales contributions de cette thèse ainsi que les perspectives de recherche.

6.1 Contributions

Nous nous sommes intéressés dans ce travail aux systèmes de Publication/Souscription pour la syndication Web. Sur le Web, une grande quantité d'informations est publiée et un grand nombre d'utilisateurs est intéressé par ces informations. Afin de garantir une diffusion efficace des informations publiées, les sites Web utilisent des flux de données sur leurs pages auxquels les utilisateurs s'abonnent et seront notifiés quand de nouvelles informations sont publiées. Toutefois, la syndication Web présente des limites devant le passage à l'échelle en quantité d'informations (milliers d'événements/minute) et en nombre de souscriptions (millions). D'où le besoin d'un système de notifications efficace permettant à l'utilisateur d'exprimer ses intérêts pour garantir une meilleure satisfaction au niveau de la quantité et de la qualité des informations reçues ainsi qu'au niveau du délai de réception.

Dans cette thèse, nous avons proposé un système de Publication/Souscription basé sur le contenu (*Content-based Publish/Subscribe System*) pour la syndication Web. Pour une recherche rapide des souscriptions satisfaites, nous avons proposé de les indexer en mémoire permettant un traitement à la volée des items publiés. Afin de mettre en place un système efficace qui prend en considération les caractéristiques du Web 2.0, nous avons répondu aux principales questions qui se posent pour le réaliser :

Quelles sont les principales caractéristiques de la syndication Web ?

Dans le but de proposer un index adapté au comportement des utilisateurs et à la nature des données du Web, nous avons présenté les résultats des recherches faites sur un jeu réel de 10,7 millions d'items publiés par 8 155 flux durant 8 mois à partir de mars 2010.

Nous avons proposé une classification des flux RSS selon leur taux de publication ainsi que leur variation au cours du temps, ce qui a permis de montrer que seulement 8% des flux sont productifs ($>$ 10 items/jour) et que le nombre moyen d'items publiés par flux est de 3,59 items/jour.

Afin de définir précisement le type d'indexation appropriée, nous avons étudié la taille des items publiés par ces flux ainsi que leur structure et le taux de réplication entre flux. Nous avons pu montrer que les balises titre et description sont présentes dans la majorité des items, et que les items ont une taille moyenne de 52 termes (équivalent à 36 termes distincts) qui est comprise entre celles des requêtes du Web (2-4 termes) ou bien les *tweets* (maximum 15 termes), et celles des pages Web (450-500 termes). Nous avons observé une réplication négligeable des items entre les flux de 0,41%. Ces caractérisations nous ont permis d'effectuer des simulations précises sur le traitement des items dans nos index. Elles nous ont également facilité la compréhension des comportements du filtrage par nouveauté et diversité.

Nous nous sommes également intéressés au vocabulaire à indexer qui est un paramètre très important pour dimensionner nos structures. Une étude détaillée du vocabulaire des items anglais a été présentée. Elle concerne l'évolution de sa taille, la distribution des fréquences d'occurrences de ses termes et l'évolution de leur rang. Nous avons caractérisé l'évolution de la taille du vocabulaire (1,5M termes) par la loi de *Heap* ainsi que la distribution des fréquences de leur occurrences par une distribution *Stretched Exponential* (utilisée pour la première fois dans ce domaine). Pour finir, nous avons trouvé que les rangs des termes les plus fréquents ne varient pas durant notre période d'acquisition. Ces études de variation du vocabulaire ont été effectuées pour nous permettre de prévoir l'évolutivité de nos structures, en particulier sur la taille du vocabulaire et sur la variation des rangs des termes.

Quelle structure d'indexation est la plus appropriée pour le Pub/Sub sur des mots-clés ?

Nous avons adapté trois structures d'indexation en mémoire centrale pour la syndication Web. Les deux premières sont basées sur les listes inverses (*CIL* et *RIL*) et la troisième sur les *Tries* (*POT*). Nous avons présenté leur algorithme de notification de souscriptions satisfaites par les items produits par les flux RSS. Les souscriptions considérées sont des ensembles de mots-clés qui seront notifiées lorsque l'ensemble de ces mots se trouve présent dans un item du flux RSS.

D'après les résultats de notre modélisation, nous avons remarqué que nous obtenons un gain important en nombre de nœuds pour les *Tries* par rapport aux *listes inverses*. Ce gain diminue avec de larges souscriptions où la factorisation devient moins importante. L'ordre des termes n'a pas d'impact significatif sur les tailles des *Tries* ainsi que le nombre de nœuds visités au cours du traitement d'un item. La distribution des fréquences d'occurrences des termes influe plus sur le nombre de nœuds visités que sur la taille des index.

Concernant la taille en espace mémoire utilisé, elle augmente linéairement en fonction de la taille du vocabulaire et du nombre de souscriptions. Toutefois, le POT utilise un espace mémoire quatre fois plus grand que celui utilisé par le CIL ou le RIL, dû à son implantation et malgré le gain de la factorisation. Concernant le temps de traitement d'un item, celui-ci devient plus important avec de larges vocabulaires car la probabilité de trouver ses termes parmi les termes des souscriptions augmente. En augmentant le nombre de souscriptions, le temps de traitement augmente linéairement pour les trois index. En effet, les listes des souscriptions sont plus longues à traiter pour le CIL et le RIL, alors q'uil y a plus de chemins à parcourir dans le POT car plus de combinaisons possibles de souscriptions.

Nous pouvons donc conclure que le POT reste, en temps de traitement, l'index le plus intéressant car il bénéficie de la factorisation des termes. Toutefois, il reste un index gourmand en mémoire à cause de l'implantation de la strcuture arborescente de ses nœuds contrairement au CIL. Le RIL peut être considéré comme un bon compromis entre le POT et le CIL en espace mémoire et temps de traitement.

Comment estimer l'espace mémoire des index et le temps de traitement des items ?

Nous avons détaillé les modèles analytiques pour estimer l'espace mémoire utilisé par ces structures (CIL, RIL et ROT) ainsi que le temps de traitement des items. Ces modèles prennent en considération la distribution des fréquences des tailles de souscriptions ainsi que celles des occurrences des termes. Pour chaque terme, il faut calculer sa probabilité d'apparition dans une souscription afin d'estimer la taille de sa liste inverse, et pour le cas du *Trie*, le nombre de nœuds.

Que faire des souscriptions jamais notifiées ?

Au regard de la distribution des occurrences des termes du vocabulaire du Web, un grand nombre de souscriptions (en particulier les plus larges) ne sont pas satisfaites par la définition du *broad match* car il faut que tous les termes de la souscription apparaissent dans l'item. De plus, une large souscription a une plus grande probabilité d'avoir des termes peu fréquents, et donc de ne jamais être satisfaite.

Nous avons adapté les trois index étudiés pour une satisfaction partielle des souscriptions. Dans cette approche, l'inclusion totale de la souscription dans l'item n'est pas exigée, il suffit que l'item contienne les termes les plus significatifs de la souscription. Ainsi, à chaque terme de la souscription lui est associé un poids. Si la somme des poids des termes de la souscription apparaissant dans l'item dépasse un seuil de satisfaction donné, alors l'item sera notifié. Nous avons utilisé le TDV (*Term Discrimination Value*) pour la pondération des termes. Cette approche mesure l'importance du terme comme étant sa capacité de distinguer entre les items et ne se base pas sur les fréquences des termes dans les items ou les souscriptions, comme TF-IDF. Pour cette raison, nous l'avons trouvée plus appropriée pour mesurer les poids des termes pour nos items et souscriptions.

Le *RIL* et le *POT* ne permettent pas de stocker des informations concernant le seuil de satisfaction de la souscription. En fait, dans le *RIL* l'absence du terme clé de la souscription dans les termes de l'item, empêche sa notification même s'il existe un sous-ensemble de ses termes qui satisfont son seuil de satisfaction. Dans *POT*, les chemins communs entre les souscriptions dus à la factorisation ne permettent pas d'attribuer des poids aux termes pour chaque souscription. Pour chercher les souscriptions partiellement satisfaites, nous avons proposé d'indexer pour chaque souscription tous les sous-ensembles de ses termes dont le score de satisfaction est supérieure au seuil de satisfaction.

L'évaluation expérimentale de nos index a montré que CIL_p, l'extension du *CIL* pour la satisfaction partielle, nécessite de moins de temps pour traiter un item que *CIL* comme nous n'avons plus besoin de faire la copie du compteur, par contre ses besoins en espace mémoire sont plus importants. Avec l'introduction de la satisfaction partielle dans nos index, nous avons pu comparer leurs performances avec celles d'une satisfaction totale. L'espace mémoire consommé par les index avec l'approche de décomposition des souscriptions dépasse celui de CIL_p. Toutefois, malgré le nombre effectif de souscriptions indexées dans le *RIL* et le *POT* avec la décomposition des souscriptions, ces index restent plus rapides que CIL_p. De plus, le temps de traitement dans le *POT* est presque le même que celui avec le *broad match*. Nous avons étudié l'impact du seuil de satisfaction sur les performances des index. L'approche utilisée dans CIL_p est indépendante du seuil de satisfaction, alors que pour le temps d'indexation, l'espace mémoire utilisé et le temps de traitement sont plus importants pour de petites valeurs de seuil dans les trois index avec l'approche de décomposition des souscriptions. En fait, le nombre de souscriptions indexées est bien plus grand pour de petits seuils (plus de combinaisons possibles de sous-ensembles de termes de la souscription qui satisfont le seuil), nécessitant plus d'espace mémoire et de temps d'indexation.

Comment éviter d'être submergé par la quantité d'information ?

Dans certains cas, des souscriptions sont sujettes à un trop grand nombre de notifications. En effet, les termes choisis ne sont pas suffisant pour filtrer convenablement. Ainsi, afin de réduire le nombre d'items notifiés à l'utilisateur il faut prendre en considération la qualité de l'information contenue dans les items. Nous avons alors proposé une deuxième phase

de filtrage basée sur la nouveauté et la diversité. Le filtrage est basé sur l'ensemble d'items déjà reçus par l'utilisateur (un historique par souscription). La nouveauté est filtrée item par item alors que la diversité se fait sur l'ensemble de l'historique. Nous avons adapté une fenêtre glissante en temps pour gérer l'historique des items. Les expériences montrent que les performances en espace mémoire et temps de traitement du système proposé sont indépendants de la taille de la fenêtre.

De plus, l'espace mémoire et le temps de traitement par historique diminuent avec le seuil de nouveauté, et augmente linéairement en fonction du nombre de souscriptions indexées. Le taux de filtrage par nouveauté augmente avec le seuil de nouveauté et devient plus important avec la diversité. Le taux de filtrage par nouveauté et diversité augmente également avec la taille de la fenêtre. Le filtrage dépend du nombre d'items reçus par l'utilisateur pour sa souscription, il est plus important pour les souscriptions très satisfaites par le *broad match*.

Pour finir, nous avons étudié la qualité de notre filtrage. Nous avons comparé l'ensemble d'items notifiés à l'utilisateur après le filtrage par notre approche avec celui choisi par une approche *top-k* basée sur un calcul périodique. Le calcul de l'ensemble *top-k* est fait par un algorithme incrémental *Greedy*. Nous avons calculé l'intersection entre notre résultat et celui de *top-k* et nous avons trouvé que, dans les meilleurs des cas, environ 70% des items notifiés après le filtrage par diversité sont parmi les *top-k* items. Regardant la qualité en terme de diversité (distance entre les items), les items choisis par notre filtrage sont à 0,1 près de celui choisi par le *top-k* pour la majorité des souscriptions ($\sim$ 80%).

6.2 Perspectives de recherche

Cette thèse ouvre plusieurs perspectives de recherche.

Calcul dynamique de TDV

Dans ce travail, nous avons utilisé le TDV (*Term Discrimination Value*) pour la pondération des termes des souscriptions et des items. Malgré son efficacité en terme de pertinence, cette technique de pondération est rarement utilisée à cause d'un temps de calcul important, car il faut pour chaque terme calculer la distance moyenne entre les items sans le terme. Pour nos jeux de données, nous avons calculé les valeurs de TDV des termes de notre vocabulaire anglais composé de 1,5 millions de termes, en se basant sur leur fréquence dans notre jeu global d'items acquis. Cependant nous les avons utilisées pour nos expériences sans prendre en considération l'ordre d'arrivée et l'apparition de nouveaux termes.

Nous nous intéressons à proposer un calcul efficace des nouvelles valeurs des TDV qui prend en considération le comportement du Web et la taille du vocabulaire publié. Nous avons pu voir que les termes qui apparaissent une fois ne sont pas discriminants, et

n'influent donc pas sur la densité de l'espace. Pour cela, il n'est pas intéressant de faire une mise à jour chaque fois que l'ensemble d'items est modifié. Il serait nécessaire de faire une mise à jour périodique des poids, si besoin. Afin d'éviter de refaire certains calculs, il ne faudrait pas calculer le TDV pour chaque terme du vocabulaire. En effet, seuls les termes apparaissant dans les éléments ajoutés et supprimés influent sur la densité.

Des méthodes proposées dans la littérature, permettent des calculs dynamiques pour le TDV [Crawford, 1975, Dattola, 1979]. Ces méthodes mettent à jour les poids à chaque fois qu'il y a un changement dans la base, bien que les valeurs soient presque les mêmes. De plus, un grand nombre de nouveaux termes n'influeront pas sur la densité de l'espace. Dans notre contexte, il faut alors fixer deux paramètres : l'ensemble d'items à considérer dans l'espace et les valeurs de TDV devront être mises à jour.

Concernant l'ensemble d'items, nous pouvons considérer une fenêtre glissante sur le temps pour les items et si une mise à jour est décidée seulement les items qui sont dans la fenêtre sont considérés. Pour la mise à jour il sera utile de la faire si les termes apparus dans les items influeront sur les TDV des termes du vocabulaire. Cela dépendra de leur fréquence d'apparition et le nombre d'items qui les contiennent. Une étude théorique des valeurs des TDV en fonction du nombre d'occurrences de nouveaux termes sera nécessaire pour décider si ces termes impactent les TDV pour les mettre à jour.

Recherche approchée

Nous avons proposé une approche de satisfaction partielle des souscriptions par les items publiés. Sur le Web, la majorité des moteurs de recherche intègre des notions de similarités entre les requêtes et les documents.

Une de nos perspectives de recherche est d'appliquer une formule de similarité sur les items à la volée. Comme les items sont de petites tailles comparés aux documents Web, nous pouvons espérer que cette étape de traitement ne soit pas trop coûteuse. De plus nous envisageons, au lieu de seulement traiter les termes de l'item, d'étendre l'ensemble de ses termes par des termes qui sont pertinents ou bien similaires. Cela, nous permettra de notifier les souscriptions contenant des termes qui sont similaires au contenu de l'item, par conséquent cet item correspond aux intérêts de l'utilisateur. Cette technique est appliquée pour les requêtes soumises aux moteurs de recherche afin de trouver des documents qui sont pertinents. Les termes à traiter peuvent être choisis en se basant sur une similarité sémantique en utilisant par exemple le dictionnaire *WordNet*, ou bien en se basant sur les fréquences de co-occurrences entre les termes dans les items ou dans les souscriptions (combinaison des paires de termes).

Pour garantir un traitement à la volée des items et une bonne qualité du résultat, plusieurs points importants sont à étudier. Il faut d'abord étudier quel sous-ensemble de termes de l'item peut être étendu. Les termes à étendre pourraient être choisis grâce à leur capacité à se distinguer dans les items (poids ou importance). De plus, combien doit être le score de pertinence entre le terme à étendre et l'ensemble de termes choisi comme

étant similaires. Le score de pertinence pourraît être étudié en fonction de l'existence des termes similaires et leur importance.

Ainsi, pour un traitement efficace il est utile d'indexer pour chaque terme la liste des termes qui lui sont similaires. Pour garantir le passage à l'échelle avec la taille du vocabulaire sur le Web, il faut choisir un sous-ensemble des termes à garder en mémoire. Cela peut être fait en se basant sur leur fréquence et le nombre de notifications qu'ils permettent.

Architecture distribuée

Nous comptons aussi étudier le comportement de nos index dans une architecture distribuée qui permet un passage à encore plus large l'échelle. Dans les systèmes centralisés, l'item publié est évalué avec tout l'ensemble des souscriptions indexées. Le but de l'architecture distribuée est, non seulement de permettre un passage à l'échelle en nombre de souscriptions indexées, mais surtout une répartition intelligente pour un traitement encore plus rapide des items.

Dans le cadre du *broad batch*, il faut distinguer la solution *CIL* qui sépare les souscriptions dans plusieurs listes inverses de la solution pour le *RIL* et le POT qui regroupe chaque souscription dans des ensembles restreints. Dans le cas du *CIL*, la solution évidente est de faire un découpage horizontal en répartissant la charge de traitement via les souscriptions, ainsi, à l'aide d'un système de *Consistent Hashing* nous pourrions répartir la charge de traitement sur un ensemble de traitement de manière rapide et efficace en dispersant les souscriptions. Les compteurs seraient découpés localement. Concernant les solutions *RIL* et POT , le découpage doit être vertical au niveau du vocabulaire lui-même. Il serait ainsi nécessaire de décider la répartition des termes en fonction de leur fréquence (co-occurrences des termes, similarité en se basant sur *WordNet*) et taux de notification en fonction de leur sous-arbre. Toutefois, cette solution ne permet d'intégrer les termes non présents dans le vocabulaire, il faut trouver une solution adaptative qui favorise l'évolution de la répartition de l'index de termes dans un environnement distribué.

Dans le cadre du filtrage par nouveauté et diversité, la question est bien différente. Il faut pouvoir éviter l'évaluation d'items ne générant des notifications. A l'aide d'un regroupement des souscriptions lié à leur contenu, nous pourrions indexer les souscriptions en fonction de la probabilité d'avoir des items capables de les satisfaire. Ainsi, une répartition des souscriptions permettrait de distribuer la charge de traitement global, le principal enjeux restant dans la distribution des items à évaluer pour éviter de les envoyer sur l'ensemble des serveurs, mais bien aux serveurs susceptibles d'être intéressés (générer des notifications). Un nouveau système d'indexation capable de distribuer les items en fonction des regroupements de souscriptions permettrait d'adresser ce problème.

La centralisation des traitements sur un serveur distribuant les souscriptions pose de nombreux soucis de charges de traitement, et constitue régulièrement un réel goulot d'étranglement. Une solution décentralisée serait envisageable. Les méthodes de *Consistent Ha-*

shing sont une piste intéressante qui permettront de distribuer la charge des souscriptions, toutefois le traitement d'un item sera effectué sur l'ensemble des serveurs. Cette solution très en vogue avec les bases de données NoSQL oriente notre réflexion sur une parallélisation globale des traitements plutôt que sur une segmentation de ceux-ci qui sont extrêmement dépendant de la distribution des termes et de leur co-occurrence dans les souscriptions.

Publications relatives

[1] Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cedric du Mouza, Michel Scholl and Nicolas Travers. Techniques d'indexation de souscriptions pour la Syndication Web. In Ingénierie des systèmes d'information (ISI),18(4) :33–58, 2013.

[2] Zeinab Hmedeh, Cedric du Mouza, and Nicolas Travers. An intelligent PubSub filtering system. In Bases de données avancées (BDA), Octobre 2013.

[3] Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cedric du Mouza, Michel Scholl and Nicolas Travers. Subscription Indexes for Web Syndication Systems. In Proceeding of the ACM International Conference on Extending Database Technology (EDBT), Mars 2012.

[4] Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cedric du Mouza, Michel Scholl and Nicolas Travers. Indexes Analysis for Matching Subscriptions in RSS feeds. In Bases de données avancées (BDA), Octobre 2012.

[5] Zeinab Hmedeh, Nelly Vouzoukidou, Nicolas Travers, Vassilis Christophides, Cedric du Mouza, and Michel Scholl. Characterizing Web Syndication Behavior and Content. In Proceedings of International Conference on Web Information System Engineering (WISE), Octobre 2011.

[6] Zeinab Hmedeh, Nicolas Travers, Nelly Vouzoukidou, Vassilis Christophides, Cedric du Mouza, and Michel Scholl. Everything you would like to know about RSS feeds and you are afraid to ask. In Bases de données avancées (BDA), Octobre 2011.

Bibliographie

[Abbar et al., 2013] Abbar, S., Amer-Yahia, S., Indyk, P., and Mahabadi, S. (2013). Real-time recommendation of diverse related articles. In *Proceedings of the 22nd international conference on World Wide Web (WWW)*, pages 1–12, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Agrawal et al., 2009] Agrawal, R., Gollapudi, S., Halverson, A., and Ieong, S. (2009). Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM)*, pages 5–14, New York, NY, USA. ACM.

[Aguilera et al., 1999] Aguilera, M. K., Strom, R. E., Sturman, D. C., Astley, M., and Chandra, T. D. (1999). Matching Events in a Content-Based Subscription System. In *Principles of Distributed Computing (PODC)*, pages 53–61.

[Ahmad and Kondrak, 2005] Ahmad, F. and Kondrak, G. (2005). Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 955–962, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Angel and Koudas, 2011] Angel, A. and Koudas, N. (2011). Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD)*, pages 781–792, New York, NY, USA. ACM.

[Atom, 2007] Atom (2007). Atom : The Atom Publishing Protocol. J. Gregorio, ed. and Google and B. de hOra, ed. and NewBay Software. http ://tools.ietf.org/html/rfc5023.

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Bagwell, 2000] Bagwell, P. (2000). Ideal Hash Trees. Technical report, EPFL Swtzerland.

[Beitzel et al., 2004] Beitzel, S. M., Jensen, E. C., Chowdhury, A., Grossman, D. A., and Frieder, O. (2004). Hourly analysis of a very large topically categorized web query log. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 321–328.

[Beygelzimer et al., 2006] Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine learning (ICML)*, pages 97–104, New York, NY, USA. ACM.

[Bodon, 2004] Bodon, F. (2004). Surprising Results of Trie-based FIM Algorithms. In *FIMI - ICDM Workshop*.

[Bodon, 2005] Bodon, F. (2005). A trie-based APRIORI implementation for mining frequent item sequences. In *Proceedings of the 1st international workshop on open source data mining : frequent pattern mining implementations*, pages 56–65, New York, NY, USA. ACM.

[Bookstein and Swanson, 1974] Bookstein, A. and Swanson, D. (1974). Probabilistic Models for Automatic Indexing. *Journal of the American Society for Information Science*, 25(5) :312–318.

[Cambazoglu et al., 2010] Cambazoglu, B. B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., and Zheng, Z. (2010). Early exit optimizations for additive machine learned ranking systems. In *Proceedings of International Conference on Web Search and Web Data Mining (WSDM)*, pages 411–420. ACM.

[Carbonell and Goldstein, 1998] Carbonell, J. and Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, New York, NY, USA. ACM.

[Carzaniga et al., 2001] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. (2001). Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3) :332–383.

[Carzaniga and Wolf, 2003] Carzaniga, A. and Wolf, A. L. (2003). Forwarding in a content-based network. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 163–174, New York, NY, USA. ACM.

[Chen et al., 2013] Chen, L., Cong, G., and Cao, X. (2013). An efficient query indexing mechanism for filtering geo-textual data. In *Proceedings of the 2013 ACM International Conference on Management of Data (SIGMOD)*, pages 749–760, New York, NY, USA. ACM.

[Clarke et al., 2008] Clarke, C. L., Kolla, M., Cormack, G. V., Vechtomova, O., Ashkan, A., Büttcher, S., and MacKinnon, I. (2008). Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–666, New York, NY, USA. ACM.

[Clément et al., 2001] Clément, J., Flajolet, P., and Vallée, B. (2001). Dynamical Sources in Information Theory : A General Analysis of Trie Structures. *Algorithmica*, 29(1) :307–369.

[Crawford, 1975] Crawford, R. G. (1975). *Automatic dictionary construction and updating*. PhD thesis, Université de Cornell.

[Dattola, 1979] Dattola, R. T. (1979). Use of dynamic discrimination values in a document retrieval system. In *Proceedings of the 2nd annual international ACM SIGIR*

conference on Information storage and retrieval : information implications into the eighties, pages 23–32, New York, NY, USA. ACM.

[Demers et al., 2006] Demers, A. J., Gehrke, J., Hong, M., Riedewald, M., and White, W. M. (2006). Towards expressive publish/subscribe systems. In *The 10th International Conference on Extending Database Technology (EDBT)*, pages 627–644.

[Dhillon et al., 2001] Dhillon, I. S., Fan, J., and Guan, Y. (2001). Efficient clustering of very large document collections. In *Data Mining for Science and Engineering Application*.

[Downey, 2001] Downey, A. B. (2001). The structural cause of file size distributions. In *International conference on Measurement and modeling of computer systems (SIGMETRICS)*, pages 328–329. ACM.

[Drosou and Pitoura, 2009] Drosou, M. and Pitoura, E. (2009). Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4) :49–56.

[Drosou and Pitoura, 2010] Drosou, M. and Pitoura, E. (2010). Search result diversification. *SIGMOD Record*, 39(1) :41–47.

[Drosou and Pitoura, 2012a] Drosou, M. and Pitoura, E. (2012a). Disc diversity : result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment (PVLDB)*, 6(1) :13–24.

[Drosou and Pitoura, 2012b] Drosou, M. and Pitoura, E. (2012b). Dynamic diversification of continuous data. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*, pages 216–227, New York, NY, USA. ACM.

[Drosou et al., 2009] Drosou, M., Stefanidis, K., and Pitoura, E. (2009). Preference-aware publish/subscribe delivery with diversity. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS)*, pages 1–12, New York, NY, USA. ACM.

[Elmagarmid et al., 2007] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection : A survey. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 19 :1–16.

[Fabret et al., 2001] Fabret, F., Jacobsen, H.-A., Llirbat, F., Pereira, J., Ross, K. A., and Shasha, D. (2001). Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. In *International Conference on Management of Data (SIGMOD)*, pages 115–126.

[Fabret et al., 2000] Fabret, F., Llirbat, F., Pereira, J. a., and Shasha, D. (2000). Efficient matching for content-based publish/subscribe systems. Technical report, INRIA , France.

[Fotiou et al., 2010] Fotiou, N., Marias, G., and Polyzos, G. (2010). Fighting spam in publish/subscribe networks using information ranking. In *6th EURO-NF Conference on Next Generation Internet (NGI)*, pages 1–6.

[French, 2002] French, J. C. (2002). Modeling Web Data. In *Proceedings of International ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 320–321.

[Gollapudi and Sharma, 2009] Gollapudi, S. and Sharma, A. (2009). An axiomatic approach for result diversification. In *Proceedings of the 18th international conference on World Wide Web (WWW)*, pages 381–390, New York, NY, USA. ACM.

[Gough and Smith, 1995] Gough, J. and Smith, G. (1995). Efficient recognition of events in distributed systems. In *Proceedings of 18th Australasian Computer Science Conference (ACSC)*.

[Hmedeh, 2010] Hmedeh, Z. (2010). Indexation dynamique des flux RSS. Master's thesis, Université libaniase - Stage du master réalisé au laboratoire CEDRIC/CNAM.

[Hmedeh et al., 2012a] Hmedeh, Z., Kourdounakis, H., Christophides, V., du Mouza, C., Scholl, M., and Travers, N. (2012a). Indexes Analysis for Matching Subscriptions in RSS feeds. In *Bases de données avancées (BDA)*, pages 1–20.

[Hmedeh et al., 2012b] Hmedeh, Z., Kourdounakis, H., Christophides, V., du Mouza, C., Scholl, M., and Travers, N. (2012b). Subscription indexes for web syndication systems. In *Proceeding of the ACM International Conference on Extending Database Technology (EDBT)*, pages 311–322, Berlin, Germany. ACM.

[Hmedeh et al., 2013] Hmedeh, Z., Kourdounakis, H., Christophides, V., du Mouza, C., Scholl, M., and Travers, N. (2013). Techniques d'indexation de souscriptions pour la Syndication Web. *Ingénierie des systèmes d'information (ISI)*, 18(4) :33–58.

[Hmedeh et al., 2011a] Hmedeh, Z., Travers, N., Vouzoukidou, N., Christophides, V., du Mouza, C., and Scholl, M. (2011a). Everything you would like to know about RSS feeds and you are afraid to ask. In *Bases de données avancées (BDA)*, pages 1–20.

[Hmedeh et al., 2011b] Hmedeh, Z., Vouzoukidou, N., Travers, N., Christophides, V., du Mouza, C., and Scholl, M. (2011b). Characterizing Web Syndication Behavior and Content. In *Proceedings of International Conference on Web Information System Engineering (WISE)*, pages 29–42. Springer Heidelberg.

[Hu and Chou, 2009] Hu, C.-L. and Chou, C.-K. (2009). RSS watchdog : an instant event monitor on real online news streams. In *Proceeding of the ACM conference on Information and knowledge management (CIKM)*, pages 2097–2098.

[Irmak et al., 2006] Irmak, U., Mihaylov, S., Suel, T., Ganguly, S., and Izmailov, R. (2006). Efficient Query Subscription Processing for Prospective Search Engines. In *Annual Technical Conference, General Track*, pages 375–380.

[Jansen and Pooch, 2001] Jansen, B. J. and Pooch, U. W. (2001). A Review of Web Searching Studies and a Framework for Future Research. *Journal of the American Society for Information Science and Technology (JASIST)*, 52(3) :235–246.

[Jansen et al., 1998] Jansen, B. J., Spink, A., Bateman, J., and Saracevic, T. (1998). Real life information retrieval : a study of user queries on the Web. *SIGIR Forum*, 32(1) :5–17.

[Kale et al., 2005] Kale, S., Hazan, E., Cao, F., and Singh, J. P. (2005). Analysis and algorithms for content-based event matching. In *International Conference on Distributed Computing Systems (ICDCS) Workshops*, pages 363–369.

[Knuth, 1973] Knuth, D. E. (1973). *The Art of Computer Programming, Volume III : Sorting and Searching*. Addison-Wesley.

[König et al., 2009] König, A. C., Church, K. W., and Markov, M. (2009). A Data Structure for Sponsored Search. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 90–101.

[Kumar and Vassilvitskii, 2010] Kumar, R. and Vassilvitskii, S. (2010). Generalized distances between rankings. In *Proceedings of International World Wide Web Conference (WWW)*, pages 571–580.

[Laherrère and Sornette, 1998] Laherrère, J. and Sornette, D. (1998). Stretched exponential distributions in nature and economy : "fat tails" with characteristic scales. *The European Physical Journal B - Condensed Matter and Complex Systems*, 2(4) :525–539.

[Lambiotte et al., 2007] Lambiotte, R., Ausloos, M., and Thelwall, M. (2007). Word statistics in Blogs and RSS feeds : Towards empirical universal evidence. *J. Informetrics*, 1(4) :277–286.

[Levering and Cutler, 2006] Levering, R. and Cutler, M. (2006). The portrait of a common HTML web page. In *Proceedings of the 2006 ACM symposium on Document engineering (DocEng)*, pages 198–204, New York, NY, USA. ACM.

[Liu et al., 2008] Liu, H., Petrovic, M., and arno Jacobsen, H. (2008). Efficient Filtering of RSS Documents on Computer Cluster. In *Proceedings of the 17th international conference on World Wide Web (WWW)*.

[Liu et al., 2005] Liu, H., Ramasubramanian, V., and Sirer, E. G. (2005). Client Behavior and Feed Characteristics of RSS, a Publish-Subscribe System for Web Micronews. In *Proceedings of the Internet Measurement Conference (ICM)*, pages 3–3.

[Liu et al., 2009] Liu, Z., Sun, P., and Chen, Y. (2009). Structured search result differentiation. *Proceedings of the VLDB Endowment*, 2(1) :313–324.

[Ma and Zhang, 2007] Ma, S. and Zhang, Q. (2007). A Study on Content and Management Style of Corporate Blogs. In *Proceedings of the 2nd international conference on Online communities and social computing (OCSC)*, pages 116–123.

[Machanavajjhala et al., 2008] Machanavajjhala, A., Vee, E., Garofalakis, M., and Shanmugasundaram, J. (2008). Scalable ranked publish/subscribe. *Proceedings of the VLDB Endowment*, 1(1) :451–462.

[Malik and Kender, 2007] Malik, H. H. and Kender, J. R. (2007). Optimizing Frequency Queries for Data Mining Applications. In *International Conference on Data Mining (ICDM)*, pages 595–600.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

[McCreadie et al., 2010] McCreadie, R. M., Macdonald, C., and Ounis, I. (2010). News article ranking : Leveraging in the wisdom of bloggers. In *Adaptivity Personalization and Fusion of Heterogeneous Information (RIAO)*, pages 40–48.

[Miller, 1995] Miller, G. A. (1995). WordNet : A Lexical Database for English. *Communications of the ACM Journal*, 38(11) :39–41.

[Milo et al., 2007] Milo, T., Zur, T., and Verbin, E. (2007). Boosting topic-based publish-subscribe systems with dynamic clustering. In *Proceedings of the 2007 ACM International Conference on Management of Data (SIGMOD)*, pages 749–760, New York, NY, USA. ACM.

[Minack et al., 2011] Minack, E., Siberski, W., and Nejdl, W. (2011). Incremental diversification for very large sets : a streaming-based approach. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, pages 585–594. ACM.

[Montemurro, 2001] Montemurro, M. A. (2001). Beyond the Zipf-Mandelbrot law in quantitative linguistics. *Physica A : Statistical Mechanics and its Applications*, 300(3-4) :567–578.

[Oita and Senellart, 2010] Oita, M. and Senellart, P. (2010). Archiving Data Objects using Web Feeds. In *Proceedings of International Web Archiving Work (IWAW)*, pages 1–9.

[Oxford, 2009] Oxford, U. (2009). RT this : OUP Dictionary Team monitors Twitter's tweets.

[Panigrahi et al., 2012] Panigrahi, D., Das Sarma, A., Aggarwal, G., and Tomkins, A. (2012). Online selection of diverse results. In *Proceedings of the fifth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 263–272, New York, NY, USA. ACM.

[Pereira et al., 2000a] Pereira, J., Fabret, F., Llirbat, F., Preotiuc-Pietro, R., Ross, K. A., and Shasha, D. (2000a). Publish/Subscribe on the Web at Extreme Speed. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, pages 627–630.

[Pereira et al., 2000b] Pereira, J., Fabret, F., Llirbat, F., Shasha, D., and Rocquencourt, I. (2000b). Efficient Matching for Web-Based Publish/subscribe Systems. In *Proceedings of the 7th International Conference on Cooperative Information Systems (CoopIS)*.

[Petrovic et al., 2005] Petrovic, M., Liu, H., and Jacobsen, H.-A. (2005). G-topss : fast filtering of graph-based metadata. In *Proceedings of the 14th international conference on World Wide Web (WWW)*, pages 539–547, New York, NY, USA. ACM.

[Pitoura and Triantafillou, 2008] Pitoura, T. and Triantafillou, P. (2008). Self-join size estimation in large-scale distributed data systems. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 764–773.

[Pripužić et al., 2008] Pripužić, K., Žarko, I. P., and Aberer, K. (2008). Top-k/w publish/subscribe : finding k most relevant publications in sliding time window w. In *Proceedings of the second international conference on Distributed event-based systems (DEBS)*, pages 127–138, New York, NY, USA. ACM.

[Qingcheng and Youmeng, 2008] Qingcheng, L. and Youmeng, L. (2008). Extracting Content from Web Pages Based on RSS. In *International Conference on Computer Science and Software Engineering (CSSE)*, pages 218–221. IEEE.

[Ramasubramanian et al., 2006] Ramasubramanian, V., Peterson, R., and Sirer, E. G. (2006). Corona : A High Performance Publish-Subscribe System for the World Wide Web. In *Networked Systems Design and Implementation (NSDI)*, pages 2–2.

[Rose et al., 2007] Rose, I., Murty, R., Pietzuch, P. R., Ledlie, J., Roussopoulos, M., and Welsh, M. (2007). Cobra : Content-based Filtering and Aggregation of Blogs and RSS Feeds. In *Networked Systems Design and Implementation (NSDI)*, pages 3–3.

[Rowstron et al., 2001] Rowstron, A. I. T., Kermarrec, A.-M., Castro, M., and Druschel, P. (2001). Scribe : The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43.

[RSS, 2003] RSS (2003). RSS 2.0 : Really Simple Syndication. Berkman Center for Internet and Society at Harvard Law School. http ://www.rssboard.org/rss-specification.

[Sadoghi and Jacobsen, 2011] Sadoghi, M. and Jacobsen, H.-A. (2011). BE-tree : an index structure to efficiently match boolean expressions over high-dimensional discrete space. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 637–648, New York, NY, USA. ACM.

[Sadoghi and Jacobsen, 2012] Sadoghi, M. and Jacobsen, H.-A. (2012). Relevance matters : Capitalizing on less (top-k matching in publish/subscribe). In *International Conference on Data Engineering (ICDE)*, pages 786–797.

[Sadoghi and Jacobsen, 2013] Sadoghi, M. and Jacobsen, H.-A. (2013). Analysis and optimization for boolean expression indexinge. In *ACM Transactions on Database Systems (TODS)*.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. S. (1975). A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11) :613–620.

[Schmidt-Maenz and Koch, 2005] Schmidt-Maenz, N. and Koch, M. (2005). Patterns in search queries. In *Data Analysis and Decision Support*, pages 122–129.

[Shen et al., 2005] Shen, Z., Aluru, S., and Tirthapura, S. (2005). Indexing for subscription covering in Publish-Subscribe systems. In *In Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems (PDCS)*, pages 328–333.

[Shraer et al., 2013] Shraer, A., Gurevich, M., Fontoura, M., and Josifovski, V. (2013). Top-k publish-subscribe for social annotation of news. *Very Large Data Bases (PVLDB)*, 6(6) :385–396.

[Sia et al., 2007] Sia, K. C., Cho, J., and Cho, H.-K. (2007). Efficient monitoring algorithm for fast news alerts. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 19 :950–961.

[Silverstein et al., 1999] Silverstein, C., Henzinger, M. R., Marais, H., and Moricz, M. (1999). Analysis of a Very Large Web Search Engine Query Log. *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 33(1) :6–12.

[Smyth and McClave, 2001] Smyth, B. and McClave, P. (2001). Similarity vs. Diversity. In *Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 347–361.

[Spearman, 1904] Spearman, C. (1904). The proof and measurement of association between two things. *The American Journal of Psychology*, 15 :72–101.

[Spink et al., 2001] Spink, A., Wolfram, D., Jansen, B. J., and Saracevic, T. (2001). Searching the Web : The Public and Their Queries. *Journal of the American Society for Information Science and Technology (JASIST)*, 52(3) :226–234.

[Thelwall et al., 2006] Thelwall, M., Prabowo, R., and Fairclough, R. (2006). Are Raw RSS Feeds Suitable for Broad Issue Scanning ? A Science Concern Case Study. *Journal of the American Society for Information Science and Technology (JASIST)*, 57(12) :1644–1654.

[van Rijsbergen et al., 1980] van Rijsbergen, C., Robertson, S., and Porter, M. (1980). New Models in Probabilistic Information Retrieval. *London : British Library Research and Development Report*, 5587.

[Vee et al., 2008] Vee, E., Srivastava, U., Shanmugasundaram, J., Bhat, P., and Amer-Yahia, S. (2008). Efficient computation of diverse query results. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering (ICDE)*, pages 228–236, Washington, DC, USA. IEEE Computer Society.

[Vouzoukidou, 2010] Vouzoukidou, N. (2010). On the statistical properties of web search queries. Technical report, ISL, ICS-FORTH, Greece.

[Walker, 1977] Walker, A. (1977). An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3 :253–256.

[Wang et al., 2004] Wang, B., Zhang, W., and Kitsuregawa, M. (2004). UB-Tree Based Efficient Predicate Index with Dimension Transform for Pub/Sub System. In *Database Systems for Advances Applications (DASFAA)*, pages 63–74.

[Whang et al., 2009] Whang, S., Shanmugasundaram, J., Vassilvitskii, S., Vee, E., Yerneni, R., and Garcia-molina, H. (2009). Indexing boolean expressions. *Very Large Data Bases (PVLDB)*, 2 :37–48.

[Williams and Zobel, 2005] Williams, H. E. and Zobel, J. (2005). Searchable words on the Web. *International Journal on Digital Libraries (JODL)*, 5(2) :99–105.

[Yan and Garcia-Molina, 1994] Yan, T. W. and Garcia-Molina, H. (1994). Index structures for selective dissemination of information under the boolean model. *ACM Transactions on Database Systems (TODS)*, 19(2) :332–364.

[Yan and Garcia-Molina, 1999] Yan, T. W. and Garcia-Molina, H. (1999). The SIFT Information Dissemination System. *ACM Transactions on Database Systems (TODS)*, 24(4) :529–565.

[Yu et al., 2009a] Yu, C., Lakshmanan, L., and Amer-Yahia, S. (2009a). It takes variety to make a world : diversification in recommender systems. In *Proceedings of the 12th*

International Conference on Extending Database Technology : Advances in Database Technology (EDBT), pages 368–378, New York, NY, USA. ACM.

[Yu et al., 2009b] Yu, C., Lakshmanan, L. V. S., and Amer-Yahia, S. (2009b). Recommendation Diversification Using Explanations. In *Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE)*, pages 1299–1302, Washington, DC, USA. IEEE Computer Society.

[Zhang et al., 2002] Zhang, Y., Callan, J., and Minka, T. (2002). Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 81–88, New York, NY, USA. ACM.

[Zien et al., 2001] Zien, J. Y., Meyer, J., Tomlin, J. A., and Liu, J. (2001). Web Query Characteristics and their Implications on Search Engines. In *Poster Proceedings of the Tenth International World Wide Web Conference (WWW)*.

[Zobel and Moffat, 2006] Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 38(2).

Zeinab HMEDEH

Indexation pour la recherche par le contenu textuel de flux RSS

Résumé

Afin de réduire l'intervalle de temps nécessaire entre la publication de l'information sur le Web et sa consultation par les utilisateurs, les sites Web reposent sur le principe de la Syndication Web. Les fournisseurs d'information diffusent les nouvelles informations à travers des flux RSS auxquels les utilisateurs intéressés peuvent s'abonner. L'objectif de la thèse est de proposer un système de notification passant à l'échelle du Web, prenant en considération le grand nombre d'utilisateurs et le débit élevé d'items. Nous proposons un index basé sur les mots-clés des requêtes utilisateurs permettant de retrouver ceux-ci dans les items des flux. Trois structures d'indexation de souscriptions sont présentées. Un modèle analytique pour estimer le temps de traitement et l'espace mémoire de chaque structure est détaillé. Nous menons une étude expérimentale approfondie de l'impact de plusieurs paramètres sur ces structures. Pour les souscriptions jamais notifiées, nous adaptons les index étudiés pour prendre en considération leur satisfaction partielle. Afin de réduire le nombre d'items reçus par l'utilisateur, nous intégrons une deuxième phase de filtrage par nouveauté et diversité considérant l'ensemble d'items déjà reçus par l'utilisateur.

Mots-clés : Syndication Web, Flux RSS, Système Publication/Souscription, Indexation, Diversité, Nouveauté.

Résumé en anglais

Based on a Publish/Subscribe paradigm, Web Syndication formats such as RSS have emerged as a popular means for timely delivery of frequently updated Web content. According to these formats, information publishers provide brief summaries of the content they deliver on the Web, while information consumers subscribe to a number of RSS feeds and get informed about newly published items. The goal of this thesis is to propose a notification system which scales on the Web. To deal with this issue, we should take into account the large number of users on the Web and the high publication rate of items. We propose a keyword-based index for user subscriptions to match it on the fly with incoming items. We study three indexing techniques for user subscriptions. We present analytical models to estimate memory requirements and matching time. We also conduct a thorough experimental evaluation to exhibit the impact of critical workload parameters on these structures. For subscriptions which are never notified, we adapt the indexes to support a partial matching between subscriptions and items. We integrate a diversity and novelty filtering step in our system in order to decrease the number of notified items for short subscriptions. This filtering is based on the set of items already received by the user.

Key-words: Web Syndication, RSS Feed, Publish/Subscribe system, Indexing, Diversity, Novelty.