THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité
Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Roxana - Gabriela HORINCAR

Pour obtenir le grade de
DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Refresh Strategies and Online Change Estimation for Highly Dynamic Web Content
Stratégies de Rafraîchissement et Estimation en Ligne de Changements pour le Contenu Web Dynamique

soutenue le 20 septembre 2012, devant le jury composé de :

M. Philippe Lamarre	Rapporteur	INSA Lyon
M. David Gross-Amblard	Rapporteur	Université de Rennes 1
Mme. Laure Berti-Equille	Examinateur	IRD, Aix-Marseille Université
M. Matthieu Cord	Examinateur	UPMC Paris 6
M. Bernd Amann	Directeur de thèse	UPMC Paris 6
M. Thierry Artières	Codirecteur de thèse	UPMC Paris 6

Motto:

Colonel Manfred von Holstein: **There is nothing a German officer cannot do.**
Captain Rumpelstoss: But... how will I learn to fly, Herr Colonel?
Col. Manfred von Holstein: The way we do everything in the German army: From the book of instructions.
[Reading from flight instruction manual]
Col. Manfred von Holstein: Step one: sit down...

[Those Magnificent Men in Their Flying Machines or How I Flew from London to Paris in 25 hours 11 minutes, 1965]

Acknowledgements

First and foremost, I would like to express my gratitude towards my two advisors, Bernd Amann and Thierry Artières who taught me the good reflexes of a researcher, how to ask the right questions and what being rigorous means. I have always admired (and secretly envied) their rigorous scientific approach, extensive knowledge and problem solving skills, but also their generosity, kindness, optimism and endless patience, for which I remain indebted. Working with them has been both a pleasure and a privilege!

Many thanks to Philippe Lamarre and David Gross-Amblard for graciously accepting to review this manuscript in spite of all the constraints, the short time and especially the month of August.

I am also grateful to Laure Berti-Equille and Matthieu Cord for the interest they have expressed with regard to this thesis and for accepting to be part of the jury.

During the years spent at LIP6 (Laboratoire d'Informatique de Paris 6), I had the chance of being surrounded by people who have inspired me through their genuine passion and hard work. Special thanks go to Bernadette Bouchon-Meunier, without whose support this thesis could not have started. I also thank Nicolas Labroche who introduced me to the DAPA (Données et Apprentissage Artificiel) community.

I am very fortunate to have benefited from the kind support, advice and wonderful work environment offered by the members of the BD (Bases de Données) team. Therefore my special thoughts go to Stéphane Gançarski, Anne Doucet, Camelia Constatin, Hubert Naacke, but also to Idrissa, Nelly, Andres and more particularly to Zeynep, Myriam and Jordi, my "PhD brothers". I thank each and every one of them for the warmth and support I have felt in their midst and for the numerous inspirational discussions.

Wholehearted thanks to all my friends in Paris, quite good and quite a few as I count proudly. And special thoughts to Alina, Consuela, Veronica and Mihai, that have been with me from the very beginnings of this French adventure. It is because of them that I love Paris.

Finally and on a more personal note, I would like to thank my mom and dad for being there for me and last but not least, Jérôme, my new family. Their unconditional love and support helped me overcome all those many bitter moments. Thanks for bearing with me!

Words are not enough. People matter - full stop. The rest comes after.

Résumé

Avec l'expansion importante d'appareils connectés à l'Internet et l'essor du Web 2.0, le contenu web se diversifie et devient de plus en plus dynamique. Afin de faciliter la diffusion de flux d'informations évolutives et souvent temporaires (news, messages, annonces), des nombreuses applications web publient les items d'informations les plus récentes dans des documents RSS ou Atom qui sont ensuite collectés et transformés par des agrégateurs RSS comme Google Reader ou Yahoo! News. Nos recherches se placent dans le contexte d'agrégation de documents RSS dynamiques et se focalisent sur l'optimisation du rafraîchissement et de l'estimation en ligne du changement de contenu RSS hautement dynamique. Nous introduisons et formalisons deux mesures qualitatives spécifiques à l'agrégation de flux RSS qui reflètent la complétude et la fraîcheur moyenne du flux d'information agrégé. Nous proposons une stratégie de rafraîchissement du "meilleur effort" qui maximise la qualité de l'agrégation par rapport aux autres approches existantes avec un nombre moyen de rafraîchissements identique. Nous présentons une analyse des caractéristiques générales de l'activité de publication des flux RSS réels en se focalisant surtout sur la dimension temporelle. Nous étudions différents modèles et méthodes d'estimation de changements d'activité et leur intégration dans les stratégies de rafraîchissement. Les méthodes présentées ont été implémentés et testés sur des données synthétiques et des flux RSS réels.

Mots clés : flux RSS, stratégie de rafraîchissement, estimation de changements en-ligne, agrégateur RSS, web dynamique, qualité de données

Abstract

With the rapidly increasing number of sources and devices connected to the Internet and the growing success of the Web 2.0 services, the online available web content is getting more and more diverse and dynamic. In order to facilitate the efficient dissemination of the evolutive and often temporary information streams (news, messages, announcements), many web applications publish their most recent information items as RSS and Atom documents which are then collected and transformed by RSS aggregators like Google Reader or Yahoo! News. Our research is placed in the context of content-based feed aggregation systems and is focused on the design of optimal refresh strategies for highly dynamic RSS feed sources. First, we introduce two quality measures specific to aggregation feeds which reflect the information completeness and average freshness of the result feeds. We propose a best-effort feed refresh strategy that achieves maximum aggregation quality compared with all other existing policies with the same average number of refreshes. We analyse the characteristics of a representative collection of real-world RSS feeds focusing on their temporal dimension. We study different online change estimation models and techniques and their integration with our refresh strategy. The presented methods have been implemented and tested against synthetic and real-world RSS feed data sets.

Keywords: RSS feed, refresh strategy, online change estimation, content-based feed aggregation, dynamic web content, data quality

Contents

Introduction

In the last twenty years, the Internet has greatly evolved from a hypertext publishing system to a platform of participation and collaboration [O'R05] among users, also referred to as *Web 2.0*. The Web 2.0 services make it easy for individuals, companies and even non-technical amateurs to create and manage their content online. In particular, many Web 2.0 applications have emerged that allow users to manage, collaborate and share their personal information on the Web. Examples include social networking sites for keeping in touch with friends, family and business partners, online galleries for photos and videos or blogs for sharing diaries or travel journals, such as Blogger [bloa] blog publishing service, Wikipedia [wika] online encyclopedia, Flickr [fli] photograph sharing, Delicious [del] bookmark sharing, Digg [dig] social news website, Facebook [fac] social network and Twitter [twi] microblogging service.

This great accessibility of the Web 2.0 applications empowers users in the creation and management of content and has led to an explosion of new online materials, also known as user generated content (UGC). In 2006 Time magazine featured UGC as the "Person of the Year" [tim], in which the person of the year was "you", meaning all of the individual online content creators who contribute to user generated media. Other examples that illustrate the general *online dynamics* and *rapid growth* include the collaborative online encyclopedia Wikipedia [wika] that reports reaching almost 8000 updates per hour for the English articles and 2000 for the French ones [wikb]. Analyzing blogosphere growth, NM Incite [nmi] tracked over 181 million blogs around the world by the end of 2011, up from 36 million only five years earlier in 2006. The free news aggregator Google News [goob] covers news content aggregated from more than 25,000 publishers [gooc] from around the world and covers about 4500 sites for the English language, each of them having its own independent publication activity. Furthermore, there are numerous services that offer real time monitoring and live updates and analysis of the stock market, domain known for its highly dynamic nature.

The rapid growth of online information sources makes it difficult for a user to stay up to date with what is new on the Web. This problem has been partially solved by the introduction of the *RSS* [rssa] and *Atom* [ato] *data formats*, that have become the de-facto standards for efficient information dissemination. They allow the users to stay informed by retrieving the latest content from the sites they are interested in. Technically speaking, an RSS feed is a standard XML document containing a list of items, the publication of

each item corresponding to a change on the site associated to that RSS feed. The number of items in this list is generally limited to the last published articles. By subscribing to RSS feeds, users save time and obtain the last modifications without having to visit the site separately and also preserve their privacy, by not needing to join the site's email newsletter. RSS is largely used for update diffusion and Figure 1 illustrates the typical type of sites RSS can be found on.

Figure 1: RSS top ten site category distribution

In order to help users access and stay up to date with the new content disseminated through the RSS feeds they are subscribed to, a number of *RSS aggregators* have recently emerged and are gaining popularity, such as Bloglines [blob], Google Reader [good], RSScache [rssc] or Technorati [tec]. Having an RSS aggregation service as the central access point to reading news, it becomes much simpler for a user to discover and manage new content from a large set of RSS source feeds.

Generally speaking, content aggregators (such as RSS feeds syndication systems, but also search engines) have two choices. They can *pull* the online available content, by proactively crawling the Web for detecting new information using a standard search engine or they can agree with the content providers on a set of protocols to *push* the content to the aggregators. Nevertheless, it is the pull protocol approach that has been widely adopted in today's Web and we briefly discuss here some of the main reasons. First, the adoption of a push-based data exchange complicates communication, by adding an extra set of rules and conventions that both the content providers and aggregators must follow. On the other hand, using the pull protocols like HTTP or XML-RPC preserves the high autonomy of both information providers and aggregators. Second, a major disadvantage of the push protocol is its lack of scalability. Using the push protocol compels the content providers to maintain user state, such as a subscriber list that might become very large in time

and therefore, hard to manage. However, the pull protocol works without needing the content providers to be aware of the subscribed aggregators. And finally, adopting a push protocol requires the existence and maintenance of a trust relationship between content providers and content aggregators, which is very difficult to manage and highly unlikely in the Web's reality. A more elaborate discussion together with related work references on pull and push protocols is presented in Section 3.1.

When considering the underlying communication protocol, from the point of view of a web server there is no distinction between RSS feeds and web pages or any other web resources. All kinds of resources have to be refreshed by using the standard pull-based HTTP protocol where changes can only be detected by explicitely contacting the content provider server. Therefore aggregators as well as search engines face the same type of challenge: deciding when is the optimum refresh time moment of each resource.

Our research work is situated in the context of the RoSeS project (Really Open Simple and Efficient Syndication) [rosa], a content-based feed aggregation system. Its goal is to define a set of web resource syndication services and tools for localizing, querying, generating, composing and personalizing RSS feeds available on the Web. A RoSeS aggregator represents an RSS feed sharing system in which users are allowed to register content-based aggregation queries defined on a set of input RSS feeds, whose results are made available to users as a newly created output RSS feed. For that, it has to periodically retrieve the newly published content on the ever changing RSS feed sources.

Placed in the context of the content-based feed aggregation systems, the main *goal* of this dissertation is to improve the aggregation quality by designing optimal *refresh strategies* and *online change estimation techniques* adapted for crawling highly dynamic RSS feed sources. Despite the apparent simplicity, feed aggregation systems have many inherent challenges:

- **Large scale.** Not only that the Web is very large, but it is also permanently evolving and so are the RSS feeds hosted on websites. Therefore, feed aggregators must support an extremely high throughput by dealing with a large number of feed content providers and consumers while seeking high aggregation quality.
- **Highly dynamic content.** As any web resource, RSS feeds evolve independently of their clients and may suddenly change their publication activity. Even if the aggregator supports large scale syndication, it could never keep up with all the dynamic changes of all the feeds.
- **Refresh decision.** In order to assure the proper functioning of an aggregator, it must use an intelligent *refresh strategy* that decides when to refresh each feed source such as to assure *aggregation quality maximization* within a *minimum cost*.
- **Limited resources.** Not only that crawlers should respect politeness policies by not imposing too much of a burden on the content providers, but they are also constrained by their own internal resource limitations, such as bandwidth, storage, memory or computing consumed resources, minimizing the overall *cost*.

- **Information loss.** Since each newly published content item removes the oldest item in an RSS feed, an aggregator that does not refresh the feed quickly enough may face *item loss*. An optimal *refresh strategy* should not miss important content and keep the *completeness* of the aggregated data at high scores.
- **Time sensitiveness.** As the result content of many *aggregation queries* applied on RSS feeds are related to real world events, the value of a piece of such information degrades itself in time. An optimal *refresh strategy* deployed on a feed aggregator should be able to retrieve quickly the newly published content, such as to keep a high level of *freshness* aggregation quality.
- **Incomplete knowledge.** The dynamic nature of the web content leads to the necessity of continually updating the publication frequency estimations, using *online estimation techniques*. Since the source publication models are updated only at the time of refresh, online estimators have to deal with *incomplete knowledge* about the data change history, not knowing exactly how often, how much and when a source produces new information items.
- **Irregular estimation intervals.** In many web applications, data sources are not refreshed in regular time intervals. The exact access moment is decided by the refresh strategy, conceived to optimize certain quality measures within a minimum cost. *Irregular refresh intervals* and *incomplete change history* make the estimation process very challenging.

Contributions

In this dissertation, we address the challenges previously mentioned when designing, implementing and evaluating a content-based feed aggregation system. The main problems presented and studied in this dissertation as well as our different contributions try to answer the following list of questions.

How a content-based feed aggregation system should be conceived and which are its main characteristics? Users of a content-based feed aggregation system want to stay informed with the latest content on the topics and from the RSS feed sources they are interested in. For that, they register different aggregation queries that reflect their interests on RSS feed sets of their choice. The results of the aggregation queries are then made available to users as newly created RSS feeds. We propose a declarative feed aggregation model and architecture for a content-based feed aggregation system with a precise semantics for feeds and aggregation queries.

How should we evaluate the quality of a feed aggregation system and which are the quality measures adapted for that? For evaluating the quality of the aggregation query results, we propose two different measures: feed completeness and window freshness. Feed completeness evaluates coverage of retrieved feed items corresponding to an aggregation query. Window freshness measured at a certain time moment reflects the

fraction of recently published feed items that have not been retrieved yet. A maximum window freshness shows that the RSS feed result of an aggregation query is up to date with all its RSS feed sources. The proposed quality metrics (feed completeness and window freshness) considered simultaneously were designed to reflect the quality of newly produced aggregation feeds within a content-based feed aggregation system.

When should the crawler of a feed aggregation system refresh the RSS feed sources and what makes a refresh strategy optimal? We propose a best effort refresh strategy based on the Lagrange multipliers [Ste91] optimization method that retrieves new postings from different feed sources and maintains an optimal level of aggregation quality, obtained with a minimum cost. Our strategy maximizes both feed completeness and window freshness, fetching new items in time and avoiding item loss. The proposed policy is supported by an extensive experimental evaluation, where its performance and robustness are tested by comparing it to other refresh strategies.

How much new information do the RSS feeds publish every day? When exactly do they publish it? Are there any patterns in their publication activity? In order to better understand the publication behavior of real RSS feeds, we propose a general characteristics analysis with a focus on the temporal dimension of real RSS feed sources, using data collected over four weeks from more than 2500 RSS feeds. We start by looking into the intensity of the feed publication activity and into their daily periodicity features. We also classify the source feeds according to three different publication shapes: peaks, uniform and waves.

How should the feed publication behavior should be modeled? What is an appropriate estimation model? How can we keep the estimation models up to date with the continually changing feed publication activities? Inspired by the observations made on the real RSS feeds publication behavior, we propose and study two different models that reflect different types of feed publication activities. Furthermore, we propose two online estimation methods that correspond to the publication models that are continually updated in order to reflect the changes in the real feed publication activities. We provide an experimental evaluation of the online estimation methods in cohesion with different refresh strategies and an analysis of their effectiveness on sources with different publication behavior was made.

Organization of the Dissertation

This rest of the dissertation is structured as follows. *Chapter 1* introduces some fundamental notions of the content-based feed aggregation domain. We describe the architecture of our aggregation system in order to better understand the proposed methods and also introduce a formal content-based feed aggregation model. *Chapter 2* defines our two aggregation oriented quality measures: feed completeness and window freshness. We also explain here the saturation problem specific to RSS feeds and give an item loss measure. In *Chapter 3* we discuss the general problem of web crawling, by introducing some key

factors and objectives usually targeted by the web crawlers that impact the web crawling policies. We then focus on the RSS feeds crawling specific problem by outlining some existing approaches proposed in research regarding feed refresh strategies. In *Chapter 4* we propose a best effort refresh strategy specially conceived for feed crawling. It maximizes aggregation quality (feed completeness and window freshness) while using a minimum cost. Experiments that test our strategy against other refresh strategies are presented at the end of the chapter.

We then briefly describe in *Chapter 5* some publication activity characteristics of real RSS feeds analyzed in the research literature, followed by an examination of web change models proposed for both web pages and RSS feeds. We also analyze different methods to estimate the web change models and focus on online estimation methods. *Chapter 6* introduces a real RSS feeds analysis with an emphasis on the temporal dimension of the publication activity. We look into publication activity, periodicity and shapes. In *Chapter 7* we propose two different ways to model the publication activity of a feed source and examine methods for updating online these publication models in order to reflect the changes in the real feed publication activities. We present an experimental evaluation of the online estimation methods, testing them in cohesion with previously introduced refresh strategies. We further discuss the results obtained for sources with different publication behaviors that show the efficiency of our proposed methods.

Finally, in the *conclusion chapter*, we resume with a summary and discussion of our work, we outline the main contributions and raise a number of issues that we regard as important for future work.

Part I

Aggregation Model

Chapter 1

Content-Based Feed Aggregation

In this chapter, we introduce the notion of content-based feed aggregation. We present the functionalities of a RSS feed aggregation system, define a formal aggregation model and semantics. Last, we define a model for content-based aggregation networks and propose a network generation method, inspired by the existing theoretical models for characterizing the Internet structure.

1.1 RSS and Atom Standards

RSS [rssa] and Atom [ato] represent standard XML based data formats for publishing frequently updated information such as news headlines, blog entries or podcasts in order to ease the syndication of the web sites contents to subscribers. Note that throughout this dissertation we usually use the term of RSS to denote both RSS and Atom standards.

Technically speaking, an RSS feed is a standard XML document containing a list of time-stamped text descriptions including links to the corresponding web pages. The size of this list is generally limited to a constant value, where the publication of a new item usually removes the oldest one in the corresponding window. From the users point of view, RSS documents are perceived as a stream of items pushed to their screen. However, when considering the underlying communication protocol, from the point of view of a web server there is no distinction between RSS feeds and other web resources. Both kinds of resources have to be refreshed by using the standard pull-based HTTP protocol where changes can only be detected by explicitely contacting the server.

A *feed* in Atom terminology (or *channel* in RSS terminology) is identied by a URL from which feed entries are fetched. A feed *document* contains a list of items (or entries) as well as additional metadata information that characterizes the feed itself, such as *feed id, title, publish date and feed categories*. Each item contains a list of elements, such as the *entry id*, the *title*, the *link* from which detailed information can be obtained, the *publication* and

the *update* date of the entry and the *entry categories*. Most of these elements are optional. An example of an RSS document is shown in Figure 1.1.

```
<?xml version='1.0' encoding='UTF-8'?>
<rss version="2.0">
<channel>
 <title>Slashdot</title>
 <link>http://slashdot.org/</link>
 <description>News for nerds, stuff that matters</description>
 <language>en-us</language>
 <pubDate>2012-02-29 01:05:11</pubDate>
 <item>
    <title>Megaupload Founder Dodges Jail Again</title>
    <link>http://yro.slashdot.org/story/12/02/29/1624254</link>
    <description>Megaupload Founder Dodges Jail Again; Wife Under Investigation...</description>
    <pubDate>2012-02-29 17:25:00</pubDate>
 </item>
</channel>
</rss>
```

Figure 1.1: A sample RSS feed

The postings are arranged in the reverse chronological order where new postings are appended in the front and old postings are pushed downwards and removed. For the majority of current implementations, an RSS document contains the most recent 10 or 15 postings. New postings are added to the feed at any time without notifying their subscribers; thus, the subscribers have to poll the RSS feeds regularly and check for updates.

1.2 Feed Aggregation System

We define a content aggregator as a system that gathers web content coming from different online information sources for subsequent reuse, such as content distribution to users to suit their needs. Feed aggregators help users access and stay up to date with the new content disseminated through RSS feeds. They reduce the time and effort needed to regularly check websites for updates, creating a unique and personalized information space. There are various examples of content aggregation systems, but in this dissertation we focus on the case of RSS feeds syndication systems.

We place our work in the context of the RoSeS project (Really Open Simple and Efficient Syndication) [rosa]. RoSeS is a *feed aggregation system* of RSS [rssa] feed sharing in which personalized feeds defined by content-based aggregation queries are delivered to users.

As shown in Figure 1.2, a RoSeS aggregator mediates the interaction between a set of *users* and the set of data *sources* to which it is subscribed to. Users define a set of *content-based aggregation queries* (introduced in Section 1.3) over a subset of the sources. Data sources constantly generate new pieces of information, called *items*, *entries* or *postings*. One data

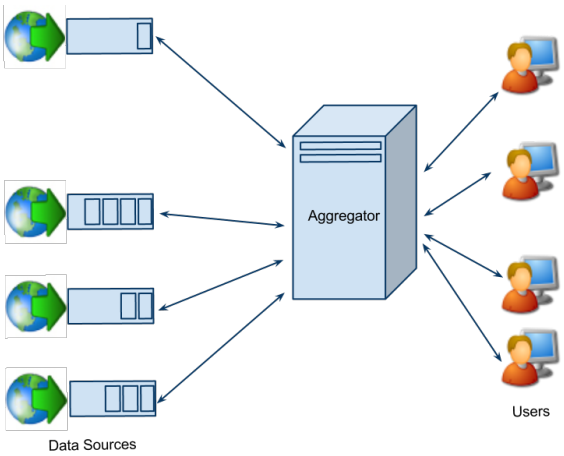


Figure 1.2: RSS Aggregator

source typically corresponds to a single (RSS [rssa] or Atom [ato]) *feed*. We say that the aggregator is *subscribed* to a data source, that it is a *subscriber* for a data source and we call a *subscription* the interaction between the aggregator and a data source. Conforming to standard web protocols, we assume a *pull*-based architecture, where the aggregator needs to periodically contact the sources in order to fetch the most recent postings from each of them. The exact refresh moment of each source is decided by the *refresh strategy* employed by the aggregator.

The result of an aggregation query applied on a set of periodically refreshed input feeds is represented by a *new feed*, typically created for two reasons. In the first case, the aggregator serves as an information diffusion and sharing tool among users, as the newly created feeds are published by the aggregator in order to be consumed by users or to serve as data sources for other aggregators. And secondly, the aggregator archives the newly created feeds. Storing feeds in databases is particularly interesting for data mining techniques and time series analysis for an increasingly important class of applications.

1.3 Content-Based Aggregation

The RoSeS system allows to formulate complex aggregation queries [TATV11] with joins and windows on feeds. Here we consider only the simplified case of *stateless continuous queries computing a filtered union of source feeds*, which represent an important subset of useful queries.

Content-based aggregation queries are defined in a declarative RSS query language on a subset of the data sources the aggregator is subscribed to. They are typically defined by users at the aggregator level, thus defining personalized RSS feeds. The result of each query is a newly created feed that can be accessed locally, be published by the aggregator to be available for other users or be stored in a database. We assume that a user might

define tens or hundreds of such aggregation queries on hundreds or thousands of source feeds.

For example, a user wants to create a feed with news about volcano eruptions in Iceland fetched from "The Guardian" and images published by "The Big Picture" on the same topic. This can easily be translated into the following aggregation query which applies a simple disjunctive filtering condition on the union of the corresponding feeds:

```
CREATE FEED IceVolEruFeed AS
RETURN "http://feeds.guardian.co.uk/theguardian/rss" AS $guardian |
       "http://www.boston.com/bigpicture/index.xml" AS $picture
WHERE $guardian[ITEM CONTAINS "iceland" OR
                ITEM CONTAINS "volcano" OR
                ITEM CONTAINS "eruption"] AND
      $picture[CATEGORY = "iceland" OR
               CATEGORY = "volcano" OR
               CATEGORY = "eruption"];
```

This query filters all items of both sources which contain at least one of the three terms "iceland", "volcano" and "eruption". The overall aggregation infrastructure is illustrated in Figure 1.3 which shows two aggregators nodes aggregating feeds from two sources.

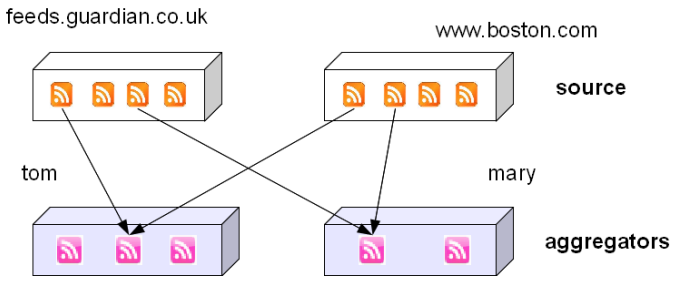


Figure 1.3: Aggregation queries

1.4 RSS Feed Aggregation Model

In this section we present feeds from several points of view. First, we define two different semantics for RSS feeds. A feed can be seen both as a continuous *stream* of all items published until a certain moment and, at the same time, as a *publication window* that holds only a finite subset of items most recently published. And last, we analyze feeds by their provenance. We distinguish between *source feeds* and *query feeds* and explain the way they are created and used. The concepts introduced and discussed in this section serve not only for a better understanding of feeds and aggregation in general, but they are also useful for the further concept definitions we present later.

1.4.1 Feed Semantics: Windows and Streams

There are mainly two different interpretations of a feed f. In the first case, we consider a feed as a continuous and possibly infinite *publication stream* of items. And in the second case, which corresponds to the standard document-based interpretation of RSS feeds, a feed f is represented by a limited number of items available in a XML document at some time instant t. Similarly as in a queue, publishing a new item consists in adding it at the beginning of the document and deleting, if necessary, the last item in the same document. The publication process is guaranteed to be done without insertion: all newly published items are appended at the beginning of the document. We will call this document a *publication window*.

Definition 1.4.1. *Item*
We consider a simplified representation of an item $i(t, K_i)$ as described by a publication date t and a set of keywords K_i that accompanies the content of the item.

Definition 1.4.2. *Publication stream*
We call the publication stream of a feed f the (possibly infinite) set of items published in f until time instant t and we denote it by $F(f, t)$.

$$F(f,t) = \{i(t_i)/t_i \leq t\}$$

More formally, we suppose that timestamps are values in a linear time dimension $(T, <)$. Then, it is possible to define the semantics of a feed as follows: let $F(f, t)$ denote the items sequence generated by f until t; then for any two time instances t and t':

- if $t = t'$, then $F(f, t) = F(f, t')$ and
- if $t < t'$, then $F(f, t) \subseteq F(f, t')$ and the set difference of the two feeds represents a possibly empty sequence of items published between t and t': $F(f, t') \setminus F(f, t) = \{i(t_i)/t < t_i \leq t'\}$.

Definition 1.4.3. *Publication window*
The publication window $A(f, t)$ of a feed f at some time instant t represents the subset of items available (or valid) at t in the publication stream $F(f, t)$. We denote by W_s the size of the publication window: $|A(f, t)| = W_s$.

$$A(f,t) = \{i(t_i)/i(t_i) \in F(f,t), |i(t_j)/t_j > t_i, i(t_j) \in F(f,t)| < W_s\}$$

Observe that the publication stream of a feed at some time instant represents the union of all publication windows up to that time instant:

$$F(f,t) = \bigcup_{t' \leq t} A(f,t')$$

1.4.2 Source and Query Feeds

In this section, we analyze and discuss feeds from their provenance point of view, as well as the way they are created and used. We distinguish between *source feeds* and *query feeds*. In the first case, we use them as input feeds consumed by the aggregator and ignore the way they are created; we call them *source feeds*. In the other case, we talk about *query feeds* when they represent the output feeds generated by an aggregator as the result of aggregation queries.

Definition 1.4.4. *Source feed*
A source feed s represents the feed generated by a source, by constantly publishing new items.

Definition 1.4.5. *Query feed*
Let q be the aggregation query applied by an aggregator on a set of source feeds $S = \{s_1, ...s_k\}$. The result of q is called a query feed and is denoted by $f = q(S)$.

As already mentioned in Section 1.3, in this dissertation we focus on the frequent case of *aggregation queries* computed as a *union of filters* over source feeds sets and has the general form: $q(S) = \bigcup_j \sigma_j(S_j)$, where $S = \cup_j S_j$.

A filter applied on a set of source feeds $\sigma_j(S_j)$ selects the items published by those source feeds that satisfy a given item predicate. Item predicates are boolean expressions (using conjunctions, disjunctions, negation) of atomic item predicates that express a condition on an item attribute. Depending on the item attribute type, atomic predicates may be applied on the publication date of an item, on the content of an item or on the set of categories that describes an item. A complex model of aggregation queries specific to the RoSeS system is presented in [TATV11].

We resume the example presented in Section 1.3 that shows how the query feed $f =$ `IceVolEruFeed` is created as a union of two filters applied on two different feed sources, $f = q(s_1, s_2) = \sigma_1(s_1) \cup \sigma_2(s_2)$. The two input feed sources come from two different websites, s_1 ="The Guardian" and s_2 ="The Big Picture". The filter applied on the first source $\sigma_1(s_1)$ selects those items that contain the keywords "iceland", "volcano" or "eruption" in their content. The second filter applied on the second source $\sigma_2(s_2)$ selects only the items that contain the same three keywords within the set of keywords associated to each item.

The items contained by the query feed $f = q(S)$ published by the aggregator come from the input source feeds in S and have passed the conditions imposed by the filters σ_j (they are said to be relevant to query q). We consider that the items are sorted on their publication date before being inserted in the query feed f. Other sorting criteria may also be considered, such as the source feed s_j from which the item comes from or some sort of item importance score.

Any query q applied on a set of source feeds introduces a *selectivity factor* with values

in $sel(q) \in [0,1]$. We consider that $sel(q) = 1$ if the aggregation query keeps all the items coming from the input source feeds (just a simple union, with no real filtering) and $sel(q) = 0$ if it filters out all items (all filters generate the empty result $\sigma_j(S_j) = \emptyset$).

The feed aggregation system introduced in Section 1.2, denoted here by n, can now be formally defined by the set of *aggregation queries* $Q(n) = \{q_1, ...q_k\}$ on $S(n) = \{s_1, ...s_m\}$. The set of source feeds of a query q and an aggregator node n is denoted by $S(q)$ and respectively by $S(n)$. Thus, $S(n) = \{S(q_i)/q_i \in Q(n)\}$.

1.5 Feed Aggregator Architecture

The general architecture of a feed aggregator node, typically a RoSeS node, is shown in Figure 1.4. It consists of three layers connected through unsynchronized item buffers: the *crawler* is in charge of crawling the collection of registered source feeds, the *query processor* executes a content-based query plan and generates for each aggregation query a query feed which is delivered by the *publisher* according to the different kinds of user subscriptions. The aggregator node transforms the publication windows of the source feeds into query feeds, by applying content-based aggregation queries.

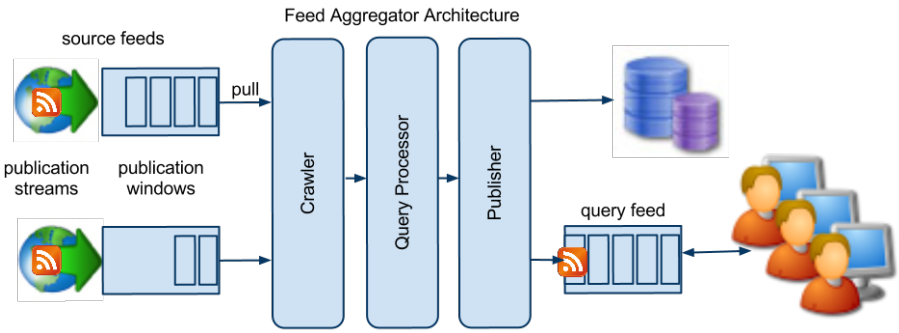


Figure 1.4: Feed aggregator node architecture

Crawler: The crawler layer periodically contacts the source feeds it is subscribed to and pulls the items available in their publication windows. The choice of the refresh moment of each source is taken by the *refresh strategy* (detailed in chapter 4) used by the aggregator. All newly retrieved feed items that were not fetched before are written into a read buffer, for the next layer to process.

Query Processor: The query processor continuously consumes the items produced by the crawler and stored in read buffers by applying aggregation queries to the corresponding items coming from different feed sources. The evaluation of the query plans are done by continuously processing the incoming feed items according to the set of aggregation queries.

The query processing approach is outside the scope of our research, but related work can be found in [TATV11].

Publisher: This module is responsible for transforming the results generated by the query processor into different output formats. The standard scenarios are to generate a new RSS query feed for each aggregation query and publish them for the user consumption or store them in a database for archiving purposes. Other output formats are also possible (SMS, email, Atom feed).

1.6 Feed Aggregation Network

The query feeds generated and published by an aggregator can also serve as input source feeds for other aggregators. If we compose several such aggregators nodes we obtain a feed aggregation network with continuous information diffusion paths.

Each node in the sharing graph can have the hybrid role of a data source and consumer in the same time, as can be seen in Figure 1.5. Nodes that have only a source role create new data and represent the entry points of the information in the system. The nodes with consumer roles are characterized by their different interests in the data produced by the sources. They can by interested by the entire information produced by a source, or only by a part of it, interests defined by different aggregation queries. As a node receives information from other source nodes, it can further distribute it into the system, having thus a hybrid producer-consumer role.

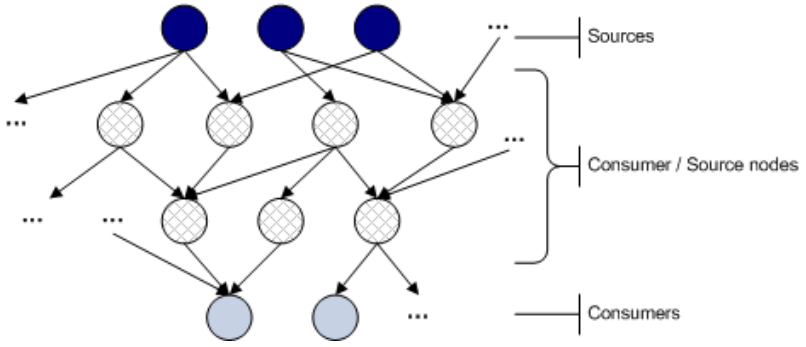


Figure 1.5: Feed aggregation graph

In the following sections, we introduce a feed aggregation subscription graph model and propose a method to construct the topology of such a network, inspired by the Internet topology.

1.6.1 Aggregation Network Model

Definition 1.6.1. *Subscription graph*
A subscription graph is defined as a directed acyclic graph $G(V, E)$, where:

- *V is a set of feeds f_i; we differentiate between two types of nodes: source feeds and query feeds that corresponds to an aggregation query q_i and*
- *E contains an edge (f_i, f_j) if f_j is defined on f_i: $\exists q_j$ such that $f_j = q_j(S)$ and $f_i \in S$.*

Observe that in this context, an aggregator n, as defined in Section 1.5, is represented here by a subset of nodes f_i from the subscription graph. The aggregator is defined as a set of aggregation queries $q_i \in Q(n)$ on $S(n)$ and each of them results in a corresponding set of query feeds $f_i = q_i(S(q_i))$, where $S(q_i) \subset S(n)$.

1.6.2 Aggregation Network Topology

We propose and analyze a syndication graph generation model designed as an internet-like topology. The original topology generation model that inspired us can be found in [FKP02, BBB+03] which is an internet-like tree topology with a power law distribution of degrees as observed in [FFF99]. Its advantage is that it is a natural and beautifully simple model of network growth involving a trade-off between geometric and network objectives.

Our goal is to create a network topology easy to introduce/design and to work with that associates potential source feeds to aggregation queries defined on them that assures the information diffusion in the graph. Differently put, we want that an aggregation query that filters on a set of keywords to be defined on source feeds that publish on (at least one of) those keywords, having in the same time a position close to the initial feed sources in the graph.

Principle As input, we have a set of nodes in the subscription graph defined as a set of source feeds $\mathbf{S} = \{s_1, ...s_m\}$ that represent the initial entry points of information in the graph. The nodes in $\mathbf{S}$ are not connected to each other. Furthermore, we complete the set of nodes of the subscription graph with a set of query feeds $\mathbf{F} = \{f_1, ...f_k\}$, each defined by an aggregation query $q_i \in Q$. The goal is to connect the node $f_i \in \mathbf{F}$ in the graph (to the nodes in $\mathbf{S}$ or to other nodes $f_j \in \mathbf{F}, f_j \neq f_i$ already connected).

Node Profile We say that every node (source feed or query feed) is described by a *node profile* that consists in a set of keywords on which the node publishes (in the case of a source feed or query feed) or in which the node is interested (in the case of a query feed).

We define a *dictionary* as the complete set of keywords $Dictionary = \{k_1, ...k_d\}$ on which the nodes can be characterized. We define the *profile of the node n* as the binary descriptor

array $Profile(n)$ of the same size d as the dictionary, defined as follows:

$$Profile(n)_i = \begin{cases} 1 & \text{if node } n \text{ publishes in the category } k_i \in Dictionary \\ 0 & \text{otherwise} \end{cases}$$

We also use the *Jaccard distance* between two node profiles $JDist(n_i, n_j) \in [0, 1]$ to compare the dissimilarity between the two of them. It is defined as:

$$JDist(n_i, n_j) = 1 - \frac{Profile(n_i) \cap Profile(n_j)}{Profile(n_i) \cup Profile(n_j)} = 1 - \frac{K_{11}}{K_{10} + K_{01} + K_{11}}$$

where K_{11} represents the number of keywords common for both profiles n_i and n_j, K_{10} the number of keywords defined for the profile of n_i, but not for n_j and K_{01} the number of keywords not defined for the profile of n_i, but defined for n_j. If K_{00} represents the number of keywords defined for neither of the two profiles, then $K_{10} + K_{01} + K_{11} + K_{00}$ represents the size of a node profile.

Connecting the Query Nodes In our model a tree is built with each source feed in $s_k \in \mathbf{S}$ as tree root. When query feed node f_i arrives, it attaches itself on one of the nodes already connected in the tree. Note that, except the root that has indegree 0, all nodes in the tree of root s_k have an indegree of 1 (only one entering edge). The first condition for the query feed node f_i to be connected in the tree of root s_k is for the two node profiles s_k and f_i to be closer than a given *radius* threshold. If the condition $JDist(s_k, f_i) \leq radius$ is met, we say that f_i in the influence radius/area of source feed s_k. Once we decided to connect f_i to the tree of root s_k, one criteria for choosing the node to which f_i connects to is the one to which it is the most similar to. Another criteria is to connect it to a centrally located node, such that its hop distance to the other nodes is minimized. Node f_i attaches itself to the already connected f_j that minimizes the formula:

$$hops(f_j) + \alpha \cdot JDist(f_j, f_i)$$

where $JDist(f_j, f_i)$ is the Jaccard distance between f_i and f_j and $hops(f_j)$ is the number of hops from the root s_k to node f_j (other measure of centrality of node f_j can also be used). α is a parameter that amplifies or reduces the influence of the similarity between node profiles.

The behavior of the model depends on the value of parameter α. If α is very small then the similarity between node profiles (Jaccard distance) is not important and the resulting network shape is similar to a star with all query feeds connected directly to the source feed root of the tree (the root has the maximum node outdegree). As the value of α grows, the topology becomes more and more clustered. On the other end, if α roughly grows bigger than the number of the nodes connected in the tree of root s_k ($\alpha > \sqrt{|tree(s_k)|}$), then the similarity between two profile nodes becomes very important and the resulting topology tends to be random and not clustered at all.

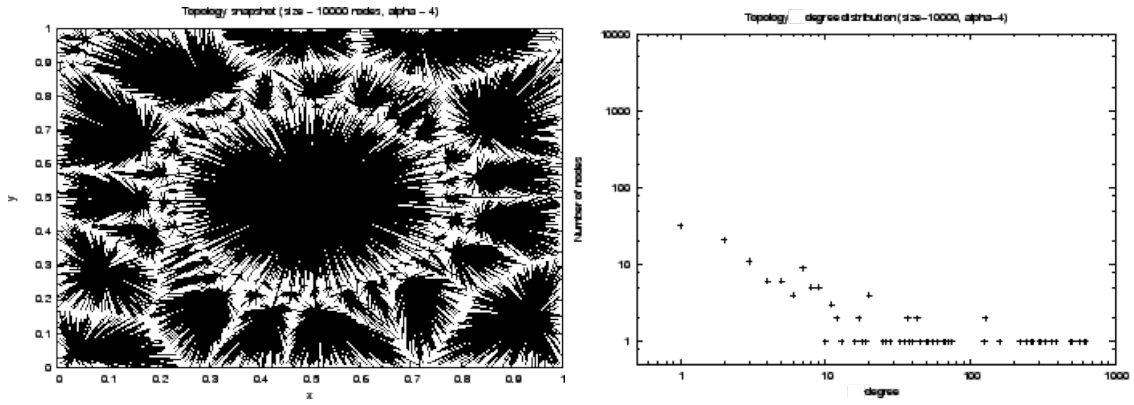
In order to illustrate the influence on α over the network shape we present in Figure 1.6 different resulting topologies obtained for 10000 nodes described in geometrical coordinates

instead of multidimensional node profiles (we use the Euclidian distance instead of the Jaccard one) and different values of α.

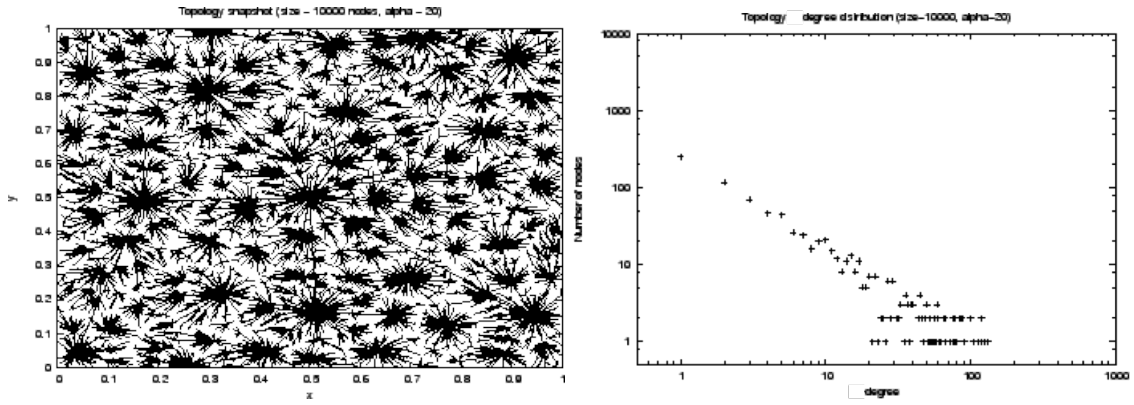
We repeat the connecting model for each source feed $s_k \in \mathbf{S}$. This way, all query feed sources $f_i \in \mathbf{F}$ has access to the data it is interested in from all sources. Since the resulting network topology depends on the order in which query feed nodes $f_i \in \mathbf{F}$ are connected in the tree of root s_k, we sort the set of query feed nodes $\mathbf{F}$ in descending order of node profiles coverage: node profiles with more values of 1 (nodes that are interested in more keywords) will be classified before those with less 1 values (that have fewer interests).

1.7 Conclusion

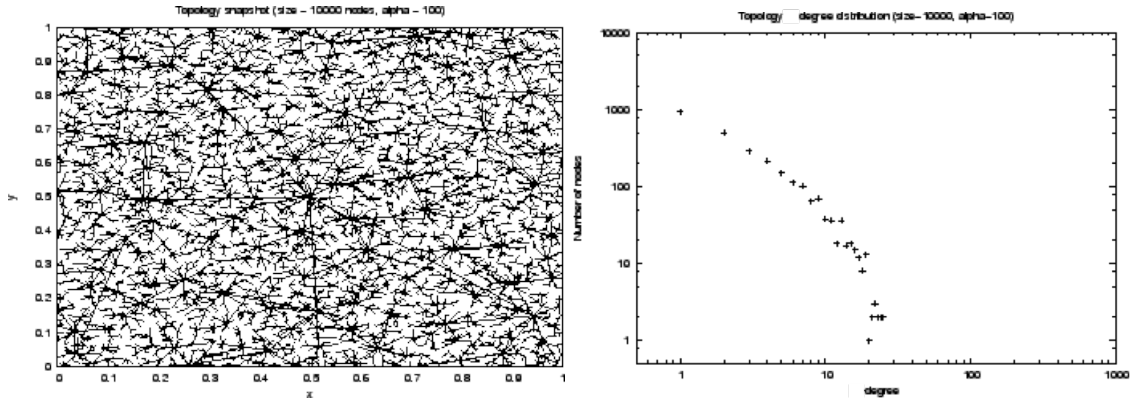
In this chapter we have presented the concept of content-based feed aggregation, a central notion for our research topic. We informally introduced a feed aggregation system and explained its functionalities and usages. We described what types of content-based aggregation queries can be registered at the aggregator level and illustrated this with an example. We also proposed a formal aggregation model, discussing feeds from several points of view. On the one side, we examine feeds semantics, by differentiating between a publication window and a publication stream. On the other side, we analyze feeds by their provenance, creation method and usage, distinguishing between source feeds and query feeds. We propose a feed aggregator architecture in three layers: crawler, query processor and publisher. As aggregators can be interconnected, we present a feed aggregation network model and discuss a topology generation method, inspired by the Internet structure. The discussed concepts are fundamental for introducing in the next chapter different specific RSS feed quality measures.



(a) $\alpha = 4$ - Topology and outdegree distribution



(b) $\alpha = 20$ - Topology and outdegree distribution



(c) $\alpha = 100$ - Topology and outdegree distribution

Figure 1.6: Network topologies for different values of α

Chapter 2

Feed Aggregation Quality Measures

Preserving the quality of the aggregated feeds represents an important challenge for the content-based feed aggregation systems. The quality of a newly created aggregated feed reflects its capacity of being fresh and complete relative to its source feeds. In other words, all items of interest published on the source feeds should be fetched as soon as possible after they are published. In this chapter, we define several types of aggregation quality measures. First, we introduce the notion of *divergence* to describe the amount of new items published by a source and not yet fetched by the aggregator. We also propose the concepts of *saturation* and *item loss* that explain what happens when a source feed is not refreshed for a long time and items are missed. We further define two feed aggregation quality measures that reflect item loss in different ways: the feed completeness and the window freshness. *Feed completeness* characterizes long-term aggregation processes and considers item loss as being definitive, where lost items are lost forever. On the other hand, *window freshness* considers aggregation process from a short-term point of view. In this case, window freshness is concerned with temporary item loss, taking into account only the missed items that still can be retrieved at refresh time.

We present a small reminder of some of the notions introduced in the previous chapter that are useful throughout this one. Let $S(n) = \{s_1, ...s_m\}$ be a set of source feeds to which the aggregator node n is subscribed to. Suppose the simple model where the aggregator node n is defined by a set of aggregation queries $Q(n) = \{q_1, ...q_k\}$ defined on $S(n)$. Let $S(q_i) = \{s_{i1}, ...s_{ij}\}, S(q_i) \subset S(n)$ be the set of sources on which query q_i is defined. Each aggregation query q_i generates a new query feed defined as $f_i = q_i(S(q_i))$. The aggregator node n refreshes periodically its sources $S(n)$, the choice of which sources to refresh and when being made based on its refresh strategy. When aggregator node n refreshes source $s_i \in S(n)$ at time t, it fetches the items available in the publication window $A(s_i, t)$ of the source feed s_i at the moment of refresh t. The purpose of this chapter is to describe how the refresh process impacts the feed aggregation quality.

2.1 Divergence

We focus on the aggregator node n, that applies an aggregation query q on the source s, where $s \in S(q)$. As introduced before, query q generates a new query feed $f = q(S(q))$, where the aggregation query q represents a union of filters over the source feeds in $S(q)$. When discussing divergence, we examine how the changes that are of interest for the aggregation query q and that occur on the source feed s are perceived and captured by the aggregator n.

We begin by formally defining the notion of divergence, as well as the measure proposed to evaluate it.

Definition 2.1.1. *Divergence*
Let T_r represent the last time moment before time t when source s has been refreshed by the aggregator. We define the divergence function $Div(s, t, T_r)$ measured at time $t > T_r$ as the total number of new items published by the source s in the time period $(T_r, t]$ that were not yet fetched by the aggregator.

Definition 2.1.2. *Divergence relevant to query q*
We similarly define the divergence function $Div(s, q, t, T_r)$ measured at time t as the total number of new items relevant to query q published by the source s in the time period $(T_r, t]$.

We suppose that the relevant items to query q are published uniformly distributed in time, such that the selectivity $sel(q)$ is considered constant. If the selectivity $sel(q)$ of the query q and the divergence $Div(s, t, T_r)$ of the source s are known, then we can estimate the divergence relevant to query q as:

$$Div(s, q, t, T_r) = sel(q) \cdot Div(s, t, T_r)$$

Observe that the values of the divergences relevant to two different queries q and q' defined on the same source s and measured at the same time instant t can be different, especially if the selectivities of the two queries are different (if $sel(q) \neq sel(q')$), then $Div(s, q, t, T_r) \neq Div(s, q', t, T_r)$.

Definition 2.1.3. *Divergence relevant to a set of queries Q*
Let Q be the set of disjoint queries defined on source s by an aggregator node: $\cap q_i = \emptyset$, where $q_i \in Q$. We define the divergence relevant to the set of queries Q measured at time t as the sum of the divergence functions relevant to all queries $q_i \in Q$.

$$Div(s, Q, t, T_r) = \sum_{q_i \in Q} Div(s, q_i, t, T_r)$$

To better understand how the divergence function behaves in time, we introduce an example in Figure 2.1. Let T_{r-1}, T_r and T_{r+1} represent the time moments at which source s is

refreshed. We consider separately the cases when the aggregation query keeps all the items coming from the input source s (when the selectivity takes its maximum value $sel(q_1) = 1$) and when it filters out some of the incoming items (when selectivity is $sel(q_2) < 1$).

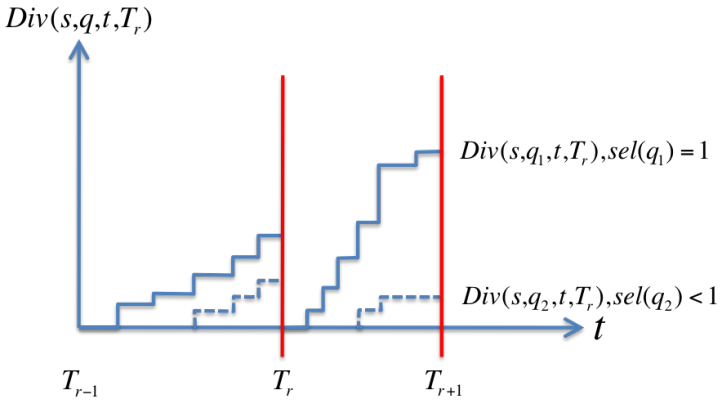


Figure 2.1: Divergence function

In the case of simple union, when the aggregator node is interested in everything source s publishes, the divergence $Div(s, q_1, t, T_r)$ represents the total number of new items published by source s since the last refresh moment T_r. Note that this case is equivalent with the simple form of the divergence function when no query is considered at all: $Div(s, q_1, t, T_r) = Div(s, t, T_r)$. This is a typical situation when the query selectivity takes the maximum value $sel(q_1) = 1$. This case is represented in Figure 2.1 by the continuous blue line.

When using the filtering query q_2, the aggregator is interested only in those items published by the source s that pass the filtering condition imposed by q_2. In this case, the query selectivity varies in the interval $sel(q_2) \in [0, 1)$ and the divergence value is smaller than in the simple union case $Div(s, q_2, t, T_r) < Div(s, q_1, t, T_r)$. This is showed in Figure 2.1 by the dotted blue line. Observe that both divergence function lines drop to 0 at refresh times T_{r-1}, T_r and T_{r+1}.

2.1.1 Saturation

Divergence, as defined previously, ignores the constraints imposed by the feed's publication window. But what happens when the number of item published by the source exceeds the size of its publication window before the aggregator gets to refresh? This is what we call *saturation*.

Definition 2.1.4. *Saturation*
Let W_s be the size of the publication window of a source s that was last refreshed at T_r. We say that source s is saturated at t if the number of newly published items $Div(s, t, T_r)$

reaches (and possibly exceeds) the size of its publication window:

$$Div(s, t, T_r) \geq W_s.$$

In consequence, we make a distinction between the total number of items published by a source since the last refresh (*stream divergence*) and the number of only those available ones (*window divergence*).

Definition 2.1.5. *Stream divergence relevant to query q*
Let $F(s,t)$ be the publication stream of source feed s. Let function $new(F(s,t))$ be the sequence of new items generated by s since its last refresh moment T_r. We define the stream divergence as:

$$Div_F(s, q, t, T_r) = |q(new(F(s,t)))|$$

Note that the divergence function introduced before (Definition 2.1.2) corresponds to the stream divergence.

Definition 2.1.6. *Window divergence relevant to query q*
Let $A(s,t)$ be the publication window of source feed s. Let function $new(A(s,t))$ return the sequence of new items published since time moment T_r and still available at source s at time moment t. We define the window divergence as:

$$Div_A(s, q, t, T_r) = |q(new(A(s,t)))|$$

Observe that stream and window divergence functions relevant to query q are equal if the source s is not yet saturated at time moment t:

$$\text{if } |new(A(s,t))| = |new(F(s,t))| < W_s, \qquad \text{then } Div_A(s, q, t, T_r) = Div_F(s, q, t, T_r)$$

Otherwise, as source s becomes saturated, stream divergence exceeds the window divergence value:

$$\text{if } |new(A(s,t))| = W_s, |new(F(s,t))| > W_s, \qquad \text{then } Div_A(s, q, t, T_r) < Div_F(s, q, t, T_r)$$

We introduce an example in Figure 2.2 in order to better understand the evolution in time of the stream and window divergence functions when different types of aggregation queries are used: a simple union q_1, with $sel(q_1) = 1$, and a filtering query q_2, with $sel(q_2) < 1$.

We consider that source s becomes saturated at time instant T_{sat}. Both the stream and window divergence functions relevant to query q_1 reach the capacity of the publication window W_s and those relevant to query q_2 reach a value estimated near $sel(q_2) \cdot W_s$. In Figure 2.2 the saturation point is marked by red lines.

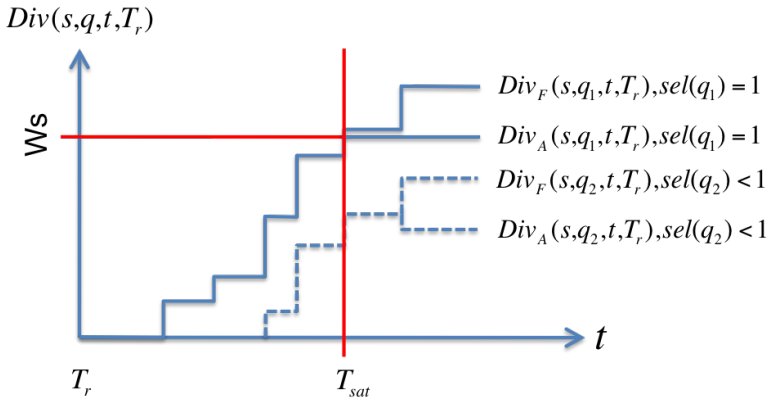


Figure 2.2: Stream and Window divergence functions

Divergence Monotonicity. In the case of the simple union q_1 (presented in Figure 2.2 by the continuous blue lines), the query selectivity has the maximum value $sel(q_1) = 1$. Both stream $Div_F(s, q_1, t, T_r)$ and window divergence $Div_A(s, q_1, t, T_r)$ functions have an identical behavior until the saturation point. Afterwards, while the stream divergence continues to increase, the window divergence reaches its maximum value $Div_A(s, q_1, t, T_r) = W_s$. Observe that for the union case q_1, both divergence functions are *monotonically increasing*.

When using the filtering query q_2 (shown in Figure 2.2 by the dotted blue line), the query selectivity varies in the interval $sel(q_2) \in [0, 1)$. As before, both stream $Div_F(s, q_2, t, T_r)$ and window divergence $Div_A(s, q_2, t, T_r)$ functions increase similarly until the saturation point. Afterwards, the arrival of new items at source s will replace items that have not been fetched yet by the aggregator. This also means that, from this moment on, the window divergence $Div_A(s, q_2, t, T_r)$ can decrease (it becomes *non monotonic*) since new items irrelevant to q_2 might replace relevant items in the publication window of s.

Divergence monotonicity is only guaranteed for the sources that have not reached saturation yet. After the saturation point, the window divergence becomes *non monotonic* and can fluctuate, taking values in the interval $[0, W_s]$. The monotonicity property of the different divergence functions is used later in chapter 4, for explaining the principle of the proposed feed refresh strategy.

2.1.2 Item loss

When putting it all together, the saturation problem comes down to the correlation between 3 functions: the *publication window size* W_s of a source s, the *publication frequency* λ of the source and the *refresh rate* r with which the aggregator n fetches the published items from that source.

For example, if a source has a publication window of size $W_s = 10$ item and it publishes new

items with the publication frequency of $\lambda = 10$ items/hour, an aggregator that refreshes that source less than once an hour $r = 1$ will lose items, as the source becomes saturated before being refreshed.

Definition 2.1.7. *Saturation coefficient*
Given the publication window size W_s, the publication frequency λ and the refresh rate r of the source s, we define the saturation coefficient $SatCoeff(n, s) \in [0, 1]$ of source s in relation with aggregator n as the percentage of items published by s and lost by n:

$$SatCoeff(n, s) = \begin{cases} 1 - \frac{W_s \cdot r}{\lambda} & \text{if } W_s \cdot r < \lambda \\ 0 & \text{otherwise} \end{cases}$$

As an example, given that the value of the publication window size is fixed to $W_s = 10$, the graph of the saturation coefficient is presented in Figure 2.3 as a function of the publication frequency λ and refresh rate r.

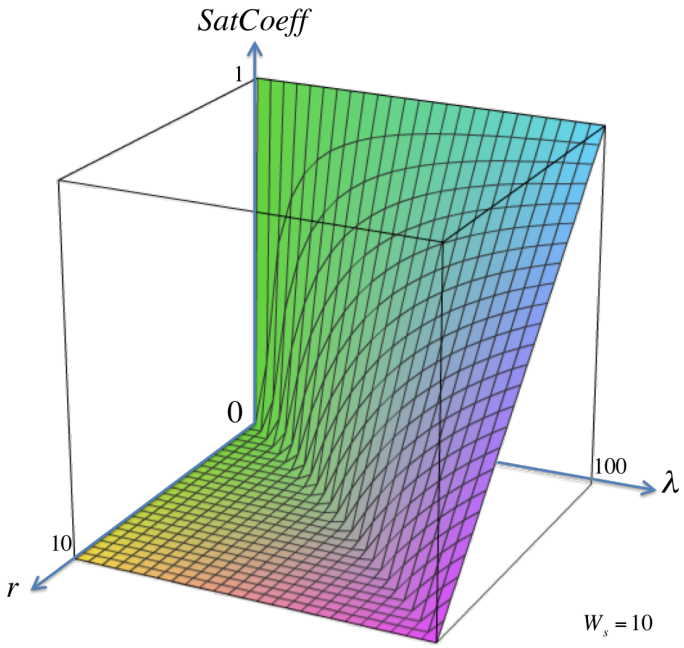


Figure 2.3: Saturation coefficient

In case the publication behavior of the sources varies in time (the publication frequency $\lambda(t) \neq const$), which is always the case in real world, the description done by the saturation coefficient is only an approximate one, useful for well understanding the saturation process.

2.2 Feed Completeness

In this section we formally define the feed completeness, a quality criteria for content-based feed aggregation services that evaluates the degree of *item loss* that occurs during a *long-term* aggregation process. This type of item loss is considered to be a *permanent* one, i.e. even if the aggregator refreshes the source, there are items that have been published by the source in the past and are no longer available to be fetched by the aggregator and the feed completeness can never take the value 1. Also, note that items are irreplaceable: if an item published by a source is lost, it can not be substituted by fetching another one from a different source.

Let $f = q(s_1, ...s_k)$ be a feed generated by some aggregation query q defined on the set of sources $S(q) = \{s_1, ...s_k\}$. And let $F(f,t)$ be the publication stream generated by feed f until time instant t. In order to formally define feed completeness, we introduce first the notion of ideal stream.

Definition 2.2.1. *Ideal stream*
We define the ideal stream $I(f,t)$ of feed f until time instant t, as depending on the nature of the feed f:

- *if $f = s$ is a source feed (as introduced in Definition 1.4.4), then the ideal stream of s, $I(s,t)$, corresponds to the stream of all items published by s until time instant t: $I(s,t) = F(s,t)$;*
- *if $f = q(s_1, ...s_k)$ is a query feed (see Definition 1.4.5), then $I(f,t) = q(I(s_1,t), ...I(s_k,t))$ corresponds to the stream of items generated by applying the aggregation query q on the streams of items generated by the sources of the query $S(q) = \{s_1, ...s_k\}$ until time instant t.*

Observe that for any feed f generated by query q, the publication stream of f is always included in the ideal feed of f:

$$F(f,t) \subseteq I(f,t)$$

Definition 2.2.2. *Feed completeness*
The feed completeness of a query feed f at some time instant t is denoted $\mathcal{C}_F(f,t) \in [0,1]$ and estimates the item loss of the aggregation process generating the feed by comparing the number of items in the publication stream $F(f,t)$ with the number of items in the ideal feed $I(f,t)$:

$$\mathcal{C}_F(f,t) = \frac{|F(f,t)|}{|I(f,t)|}$$

We say that a query feed f is *complete* when its feed completeness takes the maximum value $\mathcal{C}_F(f,t) = 1$, that is if it contains *all* the items published since the creation of the feed on all its sources $s_i \in S(q)$ on which feed $f = q(S(q))$ is defined.

2.3 Window Freshness

In this section we define the window freshness, another quality measure for content-based feed aggregation services. Window freshness evaluates *item loss*, but from a *short-term* point of view. We consider item loss as being *temporary*, i.e. all items that have not been already fetched by the aggregator can be retrieved as soon as it refreshes the source and the window freshness takes the value 1.

Window freshness takes values in the interval $[0, 1]$ and measures the quality of the aggregator publication window by the fraction of items which are available both in the publication window and in the corresponding source feed at the same time instant. It is defined in two steps. The first step defines window freshness of a query feed f generated by filter query q with respect to a single source feed $s \in S(q)$, denoted by $\mathcal{F}_W(f/s, t)$. The second step defines window freshness with respect to all sources $s_i \in S(q)$, denoted by $\mathcal{F}_W(f, t)$. Furthermore, we define window freshness with respect to all sources $s_i \in S(q)$ computed over a period of time, denoted by $\mathcal{F}_W(f)$.

Definition 2.3.1. *Window freshness with respect to a single source*
Window freshness with respect to a single source feed $s \in S(q)$ compares the number of items relevant to query q available both on the publication window of the source feed s and on the publication stream of the query feed f at time instant t, for $|q(A(s,t))| > 0$:

$$\mathcal{F}_W(f/s, t) = \frac{|F(f,t) \cap q(A(s,t))|}{|q(A(s,t))|}$$

By definition, $\mathcal{F}_W(f/s, t) = 1$ if $|q(A(s,t))| = 0$.

Definition 2.3.2. *Window freshness with respect to all sources*
The window freshness of a query feed f at time t with respect to all sources $s_i \in S(q)$ is defined as the average of all single-source freshness scores:

$$\mathcal{F}_W(f, t) = \frac{1}{|S(q)|} * \sum_{s_i \in S(q)} \mathcal{F}_W(f/s_i, t)$$

We say that a query feed f is *fresh* and its window freshness with respect to all its sources $S(q) = \{s_1, ...s_k\}$ takes the maximum value $\mathcal{F}_W(f, t) = 1$ if f contains all the items that are *available* at time instant t in the publication windows $A(s_i, t)$ of all sources $s_i \in S(q)$ and that are relevant to query q.

Definition 2.3.3. *Window freshness over a period of time*
The window freshness of a query feed f with respect to all sources $s_i \in S(q)$ computed over a period of time ΔT is defined as:

$$\mathcal{F}_W(f) = \frac{1}{\Delta T} * \int_0^{\Delta T} \mathcal{F}_W(f, t)dt$$

Observe that, as an alternative, we can define $\mathcal{F}_W(f/s,t)$, the window freshness with respect to a single source $s \in S(q)$, in function of the window divergence relevant to query q (as introduced in Definition 2.1.6), if $|q(A(s,t))| > 0$, as:

$$\mathcal{F}_W(f/s,t) = 1 - \frac{Div_A(s,q,t,T_r)}{|q(A(s,t))|}$$

Furthermore, if the aggregation query is a simple union of selectivity $sel(q_1) = 1$, then $|q_1(A(s,t))| = |A(s,t)| = W_s$ and the window freshness becomes:

$$\mathcal{F}_W(f/s,t) = 1 - \frac{|new(A(s,t))|}{W_s}$$

2.4 Conclusion

In this chapter we have presented different types of feed aggregation quality measures, notably the feed completeness and the window freshness. We discussed the divergence measure and introduced some feed specific concepts, i.e. saturation and item loss. We further analyzed feed completeness and window freshness measures that both estimate the quality of the aggregation process and of the refresh strategy used by the aggregator in terms of item loss. Feed completeness represents a refresh quality metric for long-term feed aggregation tasks (e.g. archiving), while window freshness reflects the refresh quality for short-term feed aggregation tasks (e.g. news reader).

We continue with discussing in chapter 3 several related works on the web crawling problem and we then specifically focus on RSS feeds refresh strategies. In chapter 4 we introduce and analyze a best-effort refresh strategy specially conceived for feed crawling, that maximizes aggregation quality (feed completeness and window freshness) while using a minimum cost.

Part II

Refresh Strategies

Chapter 3

Related Work

A *web crawler* (or a robot) is a system that automatically collects web resources (notably web pages) in order to create a local copy (or index). Web crawlers are a main component of web search engines and web archives (e.g. Internet Archive [inta], Internet Memory Foundation [intb]). Moreover, in the context of web data mining, crawlers are used to build web data warehouses on which statistical analysis is done. And last, web monitoring services crawl the web and notify the users of pages that match their submitted queries (e.g. Giga Alert [gig], Google Alerts [gooa]).

The web is not a centrally managed repository of information, but rather consists of big volumes of independent web content providers, each one providing their data that changes continually and independently. Furthermore, the resources allocated by the content aggregators (such as search engines or web data miners) are generally limited (bandwidth, stockage capacity, web sites politeness policy). Therefore, it is impossible for the web crawlers to capture all existent web pages and for a crawled page, to monitor it continually so to capture all changes.

This chapter discusses related work for the general problem of web crawling, with a specific focus on refresh strategies. It is organized as follows. First, we briefly present a main issue regarding the design of a web crawler, discussing the push and pull protocols. Then, we introduce some key factors and objectives that impact crawling policies. We discuss both first-time downloading strategies used for detecting and acquiring new available web content and re-downloading policies (referred here as refresh strategies) used to crawl already known web resources. Finally, we focus on the problem of RSS feeds crawling and present some refresh strategies that deal with characteristic feed issues.

3.1 Push Based Notification versus Pull Based Crawling

In a general manner, content aggregators can obtain information available on the web in two different ways. They can use a *pull* approach where they actively crawl the web to

fetch new or updated information or they receive information that suits their interests from the content providers that *push* it to the aggregators.

One of the earliest search services that adopted a push model is the Harvest system [BDH+95], but this approach did not succeed on the long run. There are a number of papers that discuss server initiated events (push), often in comparison with pull techniques. Most of them focus on distributed publish-subscribe systems [JA06, SMPD05, RPS06, RMP+07, CY08, ZSA09], multi-casting with a single publisher [AFZ97, FZ98, TV00] and AJAX applications [BMvD07]. In a more recent work, [STCR10] uses a hybrid push-pull refresh strategy for materializing users views over event feeds.

An initiative to define a push based protocol for RSS feeds has been proposed by the PubSubHubbub [pub] open protocol for distributed publish-subscribe communication on the Internet. The main purpose is to provide near-instant notifications of change updates, which would improve the typical situation where a client periodically polls the feed server at some arbitrary interval. In essence PubSubHubbub provides pushed RSS/Atom update notifications instead of requiring clients to poll whole feeds. However, the PubSubHubbub adoption heavily depends on whether both RSS feeds publishers and clients support the protocol extensions, which is rarely the case. For example, [URM+11] estimates that only 2% of their 180,000 feeds data set support the PubSubHubbub protocol.

Other researchers [OW02] have used the push protocol with a *source cooperation* approach, where data sources actively notify the remote data cache servers about their changes. They propose different types of divergence metrics and while lag is similar to our window/feed divergence measures, these metrics are conceived for a different application context and only partially reflect the particularities of RSS feeds.

Virtually all content providers and aggregators adopt the pull approach. Content providers sometime exclude all or part of their content from being crawled and may provide hints about their content, rate of change and importance. Such efforts to make web crawling more efficient by improving the underlying protocol were made by introducing the Sitemaps protocol [sit] that allows a site administrator to publish the list of available pages and their last modification date. Tradeoffs between using the classical pull approach and Sitemaps for content discovery were examined in [SS09]. Even though this protocol offers a crawler useful information about new created pages and their changes, the crawler still has to use the pull architecture in order to regularly get the information of its interest and thus, pull based refresh strategies are still needed.

Both push and pull protocols have their advantages and disadvantages. When the new content is pushed by the content providers, a big advantage is that the subscribed aggregators can receive instant update notifications as soon as new content is published. However, the problem is that the content providers must be aware of their subscribers list and manage a trust relationship with them, which is difficult and highly unlikely in the web's reality. On the other hand, when the new available content is proactively pulled from the content providers, the crawler obeys a refresh strategy that must predict the update moments and sources and that decides when the crawler fetches the newly pub-

lished content. The big advantage of the pull protocol usage is that content providers are totally unaware of their subscribers. In this case, both content providers and aggregators are independent and highly autonomous, keeping the communication protocol simple and scalable.

3.2 Crawling Web Content

Crawling policies define an *order* in which some crawler visits the web content. Crawling techniques can be compared in terms of the *factors* they consider and the *objectives* they target.

First, depending on whether the web content has been already crawled before or not, we can differentiate between:

- first-time downloading policies - that focus on the *coverage*, i.e. the fraction of desired content that the crawler acquires successfully, and
- re-downloading policies or refresh strategies - that affect *freshness*[1], i.e. the degree to which the already acquired content remain up-to-date relative to the current version of that content available online.

The tradeoff between coverage and freshness was studied in [APC03, EMT01] by combining the two objectives into a single framework that interleaves first-time downloads with re-downloads. The approach taken in [EMT01] focuses on the freshness problem, and folds in coverage by treating uncrawled pages as having zero freshness value. [APC03] focuses on ensuring coverage of important pages and in the process periodically revisits old important pages.

Second, depending on how repeated downloads are managed, we distinguish between the following design choices:

- batch crawling - where the sources to be downloaded are listed in a static crawl order list without duplicates; the crawling process may be periodically stopped and restarted in order to obtain more recent versions of the previously crawled content; this is typically associated with first-time downloading policies, and
- incremental crawling - when each source may appear multiple times in the crawl order list, so the crawling process is continuous and considered to be infinite; this can be associated both with first-time downloading and re-downloading policies, since the crawling resources are divided between downloading newly discovered content and re-downloading previously crawled content, to balance both coverage and freshness.

A detailed comparison between batch and incremental crawling architectures is presented in [CGM00b], where different properties are discussed, such as fixed or variable refresh

[1] In this chapter, the notion of *freshness* is used in a general sense; in the rest of the document, freshness appears as defining a specific RSS feed quality measure, as defined in Section 2.3.

frequency and in-place page update or page collection shadowing.

Third, some well studied factors that are widely considered in the design of a crawling policy include:

- importance - of a piece of web content (e.g. web page, RSS feed) relative to other pieces of content,
- relevance - of a piece of web content to the purpose served by the crawl, and
- content changes - or how the web content tends to evolve in time.

And last, we mention some other problems that must be considered in the crawler design, such as:

- content avoidance - which is the appropriate way a crawler should detect and avoid content that is redundant, malicious, wasteful or misleading?
- hidden or deep web - how should the web content available only by filling in HTML forms be crawled?
- personalized content - in which manner a crawler should treat web sites that offer personalized content to individual users?
- erased content - in the case a crawler runs out of storage space, how should it select the pages to retire from its repository?

In the following sections we introduce selected references that are based on some of the already discussed crawling considered factors and targeted objectives.

3.2.1 First-time Downloading Strategies

This group of techniques focuses on ordering the content for first-time download with the goal of achieving wide coverage of the available web content. Roughly put, these strategies download the web content in descending order of importance, where the importance function reflects the purpose of the crawl.

When the goal is to cover high quality content of all types, an importance score extensively used in literature is PageRank [PBMW99] and its variations. Three published empirical studies evaluate different ordering policies on real web data: [CGMP98, NW01, BYCMR05]. In [CGMP98], the authors use the indegree and the PageRank of web pages to prioritize crawling. Indegree priority favors pages with higher numbers of incoming hyperlinks. There is no consensus on prioritization by indegree: [CGMP98] finds that it works fairly well (almost as well as prioritization by PageRank), whereas [BYCMR05] finds that it performs very poorly. [NW01] uses breadth-first search, an appealing crawling strategy due to its simplicity: pages are downloaded in the order in which they are discovered, where discovery occurs via extracting links from each downloaded page. [BYCMR05] proposes a crawl policy that gives priority to sites containing a large number of discovered but

uncrawled urls. According to their empirical study, which imposed per-site politeness constraints, the proposed policy outperforms the other crawling policies (breadth-first search, indegree and PageRank) toward the end of the crawl. The reason is that it avoids the problem of being left with a few very large sites at the end, which can cause a politeness bottleneck. In [APC03] the authors propose a crawling strategy based on an algorithm called OPIC (Online Page Importance Computation), that implements an efficient online method of estimating a variant of PageRank.

Another crawling purpose that is considered is to maximize the number of pages that correspond to a set of user queries with a big search relevance impact [PO08, FCV09]. According to this argument, the goal is to crawl pages that would be viewed or clicked by search engine users, if present in the search index. More specifically, one of the most extensively studied form of crawling is the topical crawling, in which the pages relevant to a certain topic or set of topics are crawled. The intuition on which topical crawling is based on is that relevant pages tend to link to other relevant pages, either directly or through short link chains, observation that has been verified in many empirical studies, including [CGMP98, CvdBD99].

Given the large volume of available web content, there are several articles that use parallel or distributed crawling in order to maximize their throughput, having as goal the maximization of the number of collected pages, without revisiting the same pages. This is done by partitioning the url space such that each crawler machine is responsible for a subset of the urls on the web. [CGM02] present multiple architectures and strategies for conceiving parallel crawlers and study their performance. They discuss several guidelines for efficiently implementing parallel crawling, such as the number of crawling processes used and methods for coordinating different processes. During the crawling process, new page urls are extracted and added to the crawling list. When the newly discovered urls fall under the responsibility of another crawler machine, they must be forwarded and this can be done through a shared file system [HD04], peer-to-peer tcp connections [NH01] or a central coordination process [SS02, BP98].

3.2.2 Refresh Strategies

This group of techniques focuses on ordering the content for re-download, with the goal of achieving high freshness. This property measures the degree to which the already acquired content remain up-to-date relative to the current version of that content available online.

Freshness may be measured in different ways. A very popular one is *binary freshness*, also known as *obsolescence*: if a cached element is identical to its live online copy, then it is considered fresh and it is stale otherwise (freshness is the reverse of *staleness*).

One of the first studies of freshness maximization was done by [CLW98]. A key observation was that a refresh strategy that crawls a page proportionally to its change frequency (in this case, modeled by a homogeneous Poisson process) can be suboptimal. [CGM00a, CGM03a] propose an optimal refresh strategy based on the Lagrange multipliers [Ste91]

optimization method. In these works, a famous counterintuitive result was obtained: a uniform crawl strategy (where all pages are crawled with the same frequency) is superior to the proportional one (where a page is crawled proportional to its change frequency). Furthermore, their optimal crawling strategy may never revisit pages that change too often. The intuition is that the crawling resources (bandwidth) needed to keep weakly synchronized a fast changing page can be better used to keep strongly synchronized the pages that change slower for obtaining higher freshness scores. The same goal is also aimed by [WSY+02], with the difference that they no longer use a homogeneous Poisson page change model, but a quasi-deterministic page change model in which page update events are nonuniform in time and the change distribution is a priori known. The authors of [EMT01] do not make any particular assumption about the page change evolution model, but they rather use an adaptive approach that estimates and adapts the page change frequency on the fly.

Continuous freshness measures typically classify a content element as being "fresher" than others. As a first example, [CGM03a] introduces the *age* of an element as the amount of time the real and the cached copy of the element have been different. They propose a refresh strategy that minimizes age. Content-based freshness measures are introduced in [OP08], where the degree of change is reflected directly by the changes in the page content. *Fragment staleness* (they use the reverse of the freshness) is estimated by the Jaccard set distance on the component fragments of a page. In addition to characterizing a page by its update frequency, they also introduce the *longevity* of a page, as the lifetime of page fragments that appear and disappear in time. The principle of their best effort refresh strategy is also based on the Lagrange multipliers [Ste91] optimization method, but they use a variant that employs a utility function. This solution avoids computing the differential equations system of the classical Lagrange method. This same method was first presented in [OW02] in the context of cache synchronization with source cooperation and push protocol and it also represents an inspiration source for our two steps refresh strategy for RSS feeds introduced in chapter 4. For the case when the page revisitation purpose is to maintain the index of a search engine, [PO05] proposed a user-centric method that assigns weights to individual content changes, based on how this impacts the ranking of the page.

If the crawl purpose is to capture as many individual content updates as possible, then the refresh strategy goal is to maximize *completeness*. This is specific to applications such as web archiving or temporal data mining analysis. In this sense, a first algorithm was proposed in [PRC03]. In a subsequent work [PDO04], a more general algorithm is introduced that supports a flexible combination of freshness and completeness optimization.

3.3 Crawling RSS Feeds

A refresh strategy specially conceived for crawling RSS feeds is proposed in [SCC07], based on the Lagrange multipliers [Ste91] optimization method. The challenge for the

RSS aggregator is to quickly retrieve new postings in order to minimize the *delay* from the publication of the postings in an RSS feed to their appearance at the aggregator. In a subsequent work [SCH+07], a monitoring strategy based on the users previous access patterns is introduced as part of a personal information manager system. It helps a user monitor its subscribed RSS feeds by recommending him relevant articles.

Another feed monitoring approach that takes into account the feed update frequency and also its content is proposed by [RCYT08]. They introduce a profile model for the content of a feed and use their monitoring policy in order to reach their purpose: maintain dynamically channel profiles on the Web. For that, they use reinforcement learning (Boltzmann learning) to set the feed monitoring rate combined with novel content detection.

Some less elaborated order based feed refresh strategies without explicit bandwidth usage constraints are also studied in the research literature. One example is [ABP10], where RSS feeds are given different priorities to be refreshed. The RSS feed priority score depends on the expected number of newly published items and on its number of subscribers. Another example is [BGR06], that proposes different types of refresh strategies: one is the adaptive TTL policy [GS96] fitted to feed polling and another one refreshes a feed depending on the expected number of newly published items. They also explore hybrid polling strategies that switch between different strategy types. The strength of their approach resides in the elaborate change models they use. [RUM+11, URM+11] propose a minimum delay polling policy. The intuition behind it is to fetch a newly published item as soon as it is published in the feed (the refresh frequency is equal to the feed update rate).

3.4 Conclusion

In this chapter we have presented different aspects of a web crawler for different types of web content, focusing especially on web pages and RSS feeds. We started with comparing the push and pull protocols, as an important design feature of the web crawlers and continued with a discussion on different factors and objectives that impact the crawling strategies. We introduced some first-time downloading policies and then concentrated on re-downloading policies. We classified them depending on their optimization goal and scheduling techniques.

However, in the research literature there is no proposed refresh strategy specific to RSS feeds that optimizes both freshness and completeness measures and also takes into consideration the stream specific saturation process. Such a refresh strategy is introduced and discussed in the next chapter.

Chapter 4

Best Effort Refresh Strategies for RSS Feeds

The quality of the aggregated feeds generated by a content-based feed aggregation system depends entirely on the refresh strategy that is employed. Maximizing feed completeness and window freshness simultaneously is challenging in certain situations, since there is a fundamental tradeoff between them. For example, we suppose we are given a single opportunity to fetch the publication window of a feed source. If we refresh the feed early, we capture few newly published items, but we obtain a big freshness score. On the other hand, if we wait a long time before refreshing the feed, we may capture more newly published items and obtain thus a bigger completeness score, but the average freshness score is lower than in the first case. Furthermore, if we refresh the feed even later when it already got saturated and there are items that were published but are no longer available in the feed publication window, both completeness and freshness scores have even smaller values.

In this chapter, we propose a two steps best effort feed refresh strategy that privileges window freshness while there are enough bandwidth resources available so the sources do not get saturated and switches the focus on obtaining high completeness scores if the available bandwidth gets limited and many sources get saturated. We prove that our feed refresh strategy achieves maximum aggregation quality (in terms of feed completeness and window freshness) compared with all other policies that perform an equal number of refreshes. We also mention that our proposed refresh strategy does not need to pre compute a fixed refresh schedule, but it is able to assign the next time moments for refreshing the feed sources immediately prior to these time moments. This robustness property makes it compatible with online change estimation methods for modeling the feed source publication activity, as we propose in chapter 7, that update the source publication models "on the fly".

First, we present a best effort refresh strategy based on the Lagrange multipliers method and suitable for unsaturated sources that minimizes their time averaged divergence and

consequently maximizes their window freshness, spending minimum bandwidth resources. We introduce the utility function and discuss the intuition behind this best effort refresh strategy. Next, we propose a two steps best effort feed refresh strategy that takes into account the source saturation phenomenon and maximizes feed completeness. We also describe an automatically adjustment algorithm for the refresh threshold parameter. And last, we show the experimental evaluation of our two steps feed refresh strategy on simulated RSS feeds data, testing its effectiveness compared to other refresh strategies and its robustness when source publication activity is changing.

4.1 Best Effort Refresh Strategy for Unsaturated Feeds

Let $S(q) = \{s_1, ...s_k\}$ represent a set of k sources on which query q is defined. Let $Div(s_i, q, t, T_{r_i})$ be the stream divergence function of source s_i relevant to query q at time $t \geq T_{r_i}$, where T_{r_i} represents the last refresh moment of source s_i. We study the case when the source feeds have not reached yet their saturation points. Since both feed and stream divergence functions are identical up to the saturation point, in the following we choose to talk about stream divergence function to refer to either of them, until specified otherwise. We take into account bandwidth constraints and introduce b as the number of sources that can be refreshed on average per time unit. Our *goal* is to find ΔT_i, the optimal refresh periods of sources $s_i \in S(q)$, such that the total time averaged divergence is minimized. Mathematically, we can formulate our divergence minimization problem as follows:

Problem 4.1.1. *Given $Div(s_i, q, t, T_{r_i})$, find the values of ΔT_i that minimize the total time averaged divergence $\overline{Div}$*

$$\overline{Div} = \sum_{i=1}^{k} \Big(\frac{1}{\Delta T_i} \cdot \int_0^{\Delta T_i} Div(s_i, q, T_{r_i} + x, T_{r_i}) dx \Big)$$

when all ΔT_i satisfy the constraint

$$\sum_{i=1}^{k} \frac{1}{\Delta T_i} = b.$$

Solution. We use the Lagrange multipliers method [Sap06] that provides a strategy for finding the local minimum (or maximum) of a function subject to equality constraints. According to this method, solution $(\Delta T_1, ...\Delta T_k)$ satisfies condition $\nabla_{\Delta T_i, \tau} \Lambda(\Delta T_i, \tau) = 0$, where τ is the Lagrange multiplier and $\nabla_{\Delta T_i, \tau} \Lambda(\Delta T_i, \tau)$ represents the gradient of the Lagrange function $\Lambda(\Delta T_i, \tau) = \overline{Div} + \tau \cdot \Big(\sum_{i=1}^{k} \frac{1}{\Delta T_i} - b \Big)$.

Using the Lagrange multipliers method, the optimal solution has the property that there is a single constant τ for all sources s_i, such that the total time averaged divergence $\overline{Div}$

takes its minimum when all ΔT_i satisfy the following equations:

$$\frac{\partial \overline{Div_i}}{\partial \Delta T_i} = \frac{\tau}{\Delta T_i^2} \quad \text{and} \tag{4.1}$$

$$\sum_{i=1}^{k} \frac{1}{\Delta T_i} = b \tag{4.2}$$

where $\overline{Div_i} = \frac{1}{\Delta T_i} \cdot \int_0^{\Delta T_i} Div(s_i, q, T_{r_i} + x, T_{r_i})dx$.

By developing Equation 4.1, we get:

$$\Delta T_i \cdot Div(s_i, q, T_{r_i} + \Delta T_i, T_{r_i}) - \int_0^{\Delta T_i} Div(s_i, q, T_{r_i} + x, T_{r_i})dx = \tau \tag{4.3}$$

Notice that we have now a system of $(k+1)$ equations (one Equation 4.3 for each source s_i and one constraint Equation 4.2) with $(k+1)$ unknown variables $(\Delta T_1, ...\Delta T_k, \tau)$. Instead of solving these $(k+1)$ equations, we will discover the values $(\Delta T_1, ...\Delta T_k)$ for a fixed τ as follows. As the current time t advances, the optimal time moment t for refreshing s_i such that $\Delta T_i = t - T_{r_i}$ can be determined online, as the time moment at which condition in Equation 4.3 is met, for a fixed value of τ. This technique (inspired by [OW02, OP08]) is possible because the expression in Equation 4.3 *monotonically increases* (see Section 4.1.2) with t and ΔT_i.

We call τ a *refresh threshold* which controls the overall refresh rate of all sources with respect to their divergence scores. For a high value of τ, the aggregator will refresh its sources at a rather low rate and for low values of τ, it will refresh at higher rates. The actual value of τ depends on the average number of sources that the aggregator is allowed to refresh at each cycle (bandwidth constraint b) and on the update frequencies of the sources (which also impact the evolution of the source divergence functions). In case these remain constant in time, the value of τ corresponds to a constant non-negative value (see Sections 4.3 and 4.4.4).

Notice that a very important advantage of using parameter τ is that this refresh strategy is very robust, as the refresh moments can be dynamically assigned immediately prior to these time moments, which is more adapted to a dynamical environment than a pre computed fixed refresh schedule. This makes it compatible with online change estimation methods for modeling the feed source publication activity, as we propose in chapter 7, that update the source publication models "on the fly".

To resume, this optimal refresh strategy achieves minimal time averaged divergence compared with all other policies that perform an equal number of refreshes, when a source s_i is refreshed as soon as the condition in Equation 4.3 is met. We call such a strategy the *best effort refresh strategy*.

Best Effort Refresh Strategy. Let utility function $Uti(s_i, q, t, T_{r_i})$ represent the expression in Equation 4.3. Let τ be a positive constant, Div be a monotonic divergence function and T_{r_i} represent the last refresh moment of source s_i. Then, at each time instant $t \geq T_{r_i}$ refresh all the sources s_i that have an utility

$$Uti(s_i, q, t, T_{r_i}) \geq \tau \tag{4.4}$$

where utility $Uti(s_i, q, t, T_{r_i})$ is defined as follows:

$$Uti(s_i, q, t, T_{r_i}) = (t - T_{r_i}) \cdot Div(s, q, t, T_{r_i}) - \int_{T_{r_i}}^{t} Div(s_i, q, x, T_{r_i}) dx. \tag{4.5}$$

4.1.1 Utility Function

Utility function $Uti(s_i, q, t, T_{r_i})$ measured at time t, as introduced in Equation 4.5, depends on the last refresh moment T_{r_i} of source s_i and of the divergence of s_i relevant to query q during the time interval between T_{r_i} and t. The first term represents the product of the time interval since the last refresh and the current divergence. The second term captures the area under the divergence curve during the interval since the last refresh, as showed in Figure 4.1. The overall utility function $Uti(s_i, q, t, T_{r_i})$ represents the blue shaded area above the divergence curve between T_{r_i} and the current time moment t.

We represent in Figure 4.1 two examples of utility computed for two stream divergence functions. For two different sources s_1 and s_2, we show the evolution of the stream divergences $Div_F(s_1, q, t, T_{r_1})$ and $Div_F(s_2, q, t, T_{r_2})$ relevant to the same query q. The horizontal axis represents time and we assume that both sources were last refreshed at the same moment in time $T_{r_1} = T_{r_2} = T_r$.

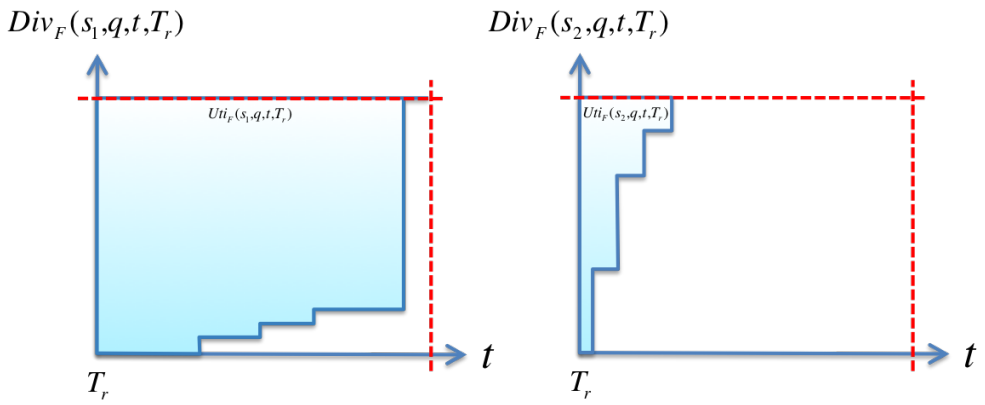


Figure 4.1: Utility and stream divergence for s_1 and s_2

Both sources reach the same stream divergence value during the same time interval, between T_r and t, but with different time evolutions. Source s_1 published few items until recently and then suddenly published many items in a small interval of time. Source

s_2 published many items immediately after its last refresh moment, but has not had a significant publication activity since then.

In Figure 4.1 source s_1 diverged slowly after its last refresh and it underwent a significant change only recently. On the other hand, source s_2 diverged very quickly after its last refresh. Since both sources have the same divergence value at time t, refreshing either one of them at t will fetch the same number of newly published items from both of them. Hence, we obtain the same gain in terms of feed completeness. Assuming it is likely that both sources will have the same behavior after the next refresh in the future, refreshing the source s_1 with the highest utility function implies obtaining a more long term benefit in terms of divergence reduction compared with refreshing s_2. Differently put, we prefer to refresh the source that is the most likely to produce a minimal time averaged divergence score in the future. Moreover, divergence minimization generates window freshness maximization, as the two measures can be formulated one depending on the other, as shown in Section 2.3.

Our conclusion is similar to that found for synchronizing web pages in [CGM03a], that given a limited refresh bandwidth, we should penalize the sources that change too often in order to keep a high freshness score. The intuition is that refresh bandwidth is put at better use if we keep strongly synchronized a source that diverges slowly than to keep weakly synchronized a fast changing one. Furthermore, our conclusion also matches the one of [SCC07] who proposes a monitoring policy for RSS feeds that minimizes delay. As in our findings, they suggest that the retrieval time should be scheduled right after a period of high publication activity.

4.1.2 Utility Monotonicity

As shown in Section 4.1, if the utility function increases monotonically as time increases, there is a single time moment t when the utility value reaches the refresh threshold value τ and it can be determined online. If we compute the time derivative of the utility function as it is defined in Equation 4.5, we get:

$$\frac{\partial Uti(s_i, q, t, T_{r_i})}{\partial t} = (t - T_{r_i}) \cdot \frac{\partial Div(s, q, t, T_{r_i})}{\partial t}.$$

Since the stream divergence semantics that we use has the property that it is a monotonically increasing function, as discussed in Section 2.1.1, its time derivative is nonnegative ($\partial Div(s, q, t, T_{r_i})/\partial t \geq 0$). Therefore, the utility time derivative is also nonnegative ($\partial Uti(s_i, q, t, T_{r_i})/\partial t \geq 0$) and thus, the utility function increases monotonically with time.

4.2 2Steps Best Effort Refresh Strategy for Saturated and Unsaturated Feeds

The best effort refresh strategy proposed in Section 4.1 stands for monotonic divergence functions, as it is the case of the stream divergence semantics (see the monotonicity discussion presented in Section 2.1.1). The challenge we encounter is that under limited bandwidth it is not possible to attain stream semantics and we must take into consideration the saturation phenomenon (defined in Section 2.1.1).

Maintaining stream divergence semantics becomes unrealistic and the usage of the window divergence semantics imposes itself. As discussed in Section 2.1.1, since window divergence is increasing monotonically until the source gets saturated and becomes non monotonic afterwards, sources which already have reached their saturation point must be handled separately, in order do minimize item loss.

2Steps Best Effort Refresh Strategy for Saturated and Unsaturated Feeds. Based on these observations, we define a two steps refresh strategy as described in the Algorithm 4.1. The first step will refresh the top-b *saturated* sources that have maximum positive window divergence $Div_A(s_i, q, t, T_{r_i}) > 0$. If the saturated sources are not refreshed right away and they continue publishing, there is highly possible that (more) items get lost. Therefore, this criteria ensures that permanent items loss is minimized and, implicitly, that the feed completeness is maximized. The following step handles the rest of the sources (that are not saturated) by refreshing them based on the best effort refresh strategy introduced in Section 4.1. As explained before, this refreshing criteria guarantees divergence minimization and window freshness maximization. Since the refresh threshold τ in Equation 4.4 corresponds to an average limited bandwidth of b refreshed sources, threshold τ must be adjusted to the fraction of the bandwidth available after the first step.

4.3 Refresh Threshold Adjustment

Parameter τ represents the *refresh threshold* which controls the global refresh rate of all the sources with respect to their divergence scores. The actual value of τ depends on the aggregator's bandwidth constraints b and on the publication frequencies that characterize the sources' activity.

The *refresh threshold adjustment algorithm* achieves optimal bandwidth usage by adapting the threshold τ to the available resources of a node and the publication activity of the source feeds. Given constant available resources and publication frequencies of relevant items, the value of τ converges to a non-negative value, as experimentally shown in Section 4.4.4.

Algorithm 4.1 2Steps Best Effort Refresh Strategy for Saturated and Unsaturated Feeds

Input: b, τ, $Div_A(s_i, q, t, T_{r_i})$, $Uti(s_i, q, t, T_{r_i})$
$Refreshed(t) := \emptyset$, $Saturated(t) := \emptyset$, $b_{sat} = 0$
{**FIRST Step: refresh up to** b **saturated sources ordered by window divergence** $Div_A(s_i, q, t, T_{r_i})$}
for all $s_i \in S(q)$ **do**
  if s_i is at saturation point and $Div_A(s_i, q, t, T_{r_i}) > 0$ **then**
    $Saturated(t) := Saturated(t) \cup \{s_i\}$
  end if
end for
$Refreshed(t) := top_b(Saturated(t))$ // refresh b saturated sources with the maximum window divergence $Div_A(s_i, q, t, T_{r_i})$
{**SECOND Step: refresh unsaturated sources using refresh threshold** τ}
$b_{sat} = size(Refreshed(t))$
if $b_{sat} < b$ **then**
  for all $s_i \in S(q)$ **do**
    if $Uti(s_i, q, t, T_{r_i}) \geq \frac{b}{b - b_{sat}} * \tau$ **then**
      $Refreshed(t) := Refreshed(t) \cup \{s_i\}$
    end if
  end for
end if
set new refresh threshold τ value
refresh sources in $Refreshed(t)$

However, in the context of content-based feed aggregation the publication frequency of items relevant to some query q might rapidly change in time. For example, the frequency of news items that talk about the UEFA football championship may vary depending on the real time UEFA events. Every four years, before and especially during the championship, there is constant publication activity and even information bursts on this subject, therefore the divergence and utility values of an aggregator that monitors this type of news will be high and the τ refresh threshold must adapt to these changes. Thus there is no single constant value for the refresh threshold that would be optimal all the time.

We consider a refresh threshold value τ specific to an aggregator, common to all its subscriptions. Any randomly chosen initial values of τ can be used, since the algorithm will adapt it to the optimal one. If the system parameters change, the algorithm is designed to adjust the threshold value τ dynamically so it converges to a new global best value of the refresh threshold τ.

The limited resources of an aggregator are represented by the bandwidth constraint b: the average number of sources that the aggregator is allowed to refresh in the unit time. Suppose that the aggregator has selected b' sources to refresh, then the aggregator will (i) increase the value of τ if it used more resources than available on average ($b' > b$) and it will (ii) decrease τ if the available bandwidth has been exploited below some percentage $p \in (0, 1]$ ($b' < p \cdot b$). τ doesn't change otherwise ($p \cdot b \leq b' \leq b$). Details of this *refresh threshold adjustment algorithm* are described in the Algorithm 4.2.

The value of τ is decreased by a factor $\theta \in (0, 1)$, by setting: $\tau = \theta \cdot \tau$, the decrease parameter θ controlling how aggressively the node gives priority for more feed sources to be refreshed. Similarly, the value of τ is increased with a $\Theta > 1$ factor: $\tau = \Theta \cdot \tau$. The increase parameter Θ reflects how quickly the aggregator n decides to slow down the refresh rate of the sources. Experimental results on the convergence of τ are presented in Section 4.4.4.

Algorithm 4.2 Refresh Threshold τ Adjustment

Input: b, $b' = size(Refreshed(t))$, $p \in (0, 1]$, $\theta \in (0, 1)$, $\Theta > 1$
if $b' > b$ **then**
  $\tau = \Theta \cdot \tau$ // increase τ
else if $b' < p \cdot b$ **then**
  $\tau = \theta \cdot \tau$ // decrease τ
else
  do nothing
end if

4.4 Experimental Evaluation

In this section we present the experimental evaluations of the effectiveness of our refresh strategy algorithm based on simulated RSS feeds data.

4.4.1 Parameter Settings

The simulation environment used to perform our experiments is PeerSim [pee], a Java-based engine for testing peer-to-peer protocols. We constructed a cycle-based environment with an aggregator node that applies a query q on a set of 100 feed sources $S(q) = \{s_1, ...s_{100}\}$. Each web feed s_i is generated following a homogeneous Poisson process of constant rate parameter λ_i, uniformly distributed in the interval $[0, 6.5]$.

A source s_i is characterized by a node publication profile (as described in Section 1.6.2) which consists in a set of keywords that describe the items published by s_i. The profile of a source uniformly covers on average 50% of the keywords in a dictionary of size 10. An item produced by a source has associated on average 20% of the keywords in the source publication profile. The query q of the aggregator node consists in a set of keywords/categories in which it is interested in that represent on average 40% of the keywords uniformly distributed in the dictionary. Under this setting, the selectivity factor (introduced in Section 1.4.2) of an aggregation query relative to a source profile can cover the entire range of possible values in the $[0, 1]$ interval (1 if the query selects everything produced by the source, 0 if it selects nothing).

The size of the all source publication windows is $W_s = 10$ and the aggregator node publishes all newly fetched items in which it is interested (there is no write loss). The average number of sources that can be refreshed by the aggregator during a time cycle is defined by the parameter b that varies in the interval $(0, 100]$.

Since the refresh threshold setting algorithm described in Section 4.3 dynamically adjusts the value of τ, any initial value can be used. The experiment results presented in the following section are obtained after the value of τ was adjusted during a warmup period. The values chosen for the decrease and increase parameters are $\theta = 0.95$ and $\Theta = 1.05$. We chose small values for these parameters (τ varies of 5%) since big variations of τ may trigger big oscillations in the number of refreshed sources at each refresh cycle and prevent τ from converging. τ is considered to have converged to the optimal value if the aggregator refreshes between $90\% \cdot b$ and b sources.

Hypothesis for Divergence and Utility

For using the 2Steps refresh strategy, we need to know how to compute the divergence and utility function values for each source.

Divergence Values. In a purely pull based environment it is difficult to know the divergence function values of each source at a certain time moment, without refreshing it and thus finding out the exact number of items the source have published since the last refresh. In our case, window divergence relevant to a query $Div_A(s_i, q, t, T_{r_i})$ depends not only on the total number of new items generated after the last refresh of source s_i, $Div(s_i, t, T_{r_i})$, but also on the selectivity of the query filter $sel(q)$, as shown in Section 2.1. One solution is to estimate divergence by collecting detailed statistics on the history of source publication frequency and query selectivity. Another possible idea is to introduce a push based or hybrid protocol, with the inconvenience that sources are not client agnostic anymore. In our experiments we suppose that the exact divergence values for each source and for each time moment are known as a priori information and we say that the refresh strategies use offline knowledge.

Utility Values. The utility function takes into account the evolution of divergence over time. Given that the exact divergence value $Div_A(s_i, q, t, T_{r_i})$ is known for the source s_i at time t and that s_i is updated according to a homogeneous Poisson process of constant parameter λ_i, the divergence integral in Equation 4.4 is expected to be equal to $\int_{T_{r_i}}^{t} Div_A(s_i, q, x, T_{r_i})dx = \frac{1}{2} \cdot (t - T_{r_i}) \cdot Div_A(s_i, q, t, T_{r_i})$. Under these assumptions, the utility function $Uti(s_i, q, t, T_{r_i})$ can be estimated as:

$$Uti(s_i, q, t, T_{r_i}) = \frac{1}{2} \cdot (t - T_{r_i}) \cdot Div_A(s_i, q, t, T_{r_i}).$$

We use here some simple offline estimation solutions, based on a priori known information, but we further analyze and discuss in Section 7.2 several online estimation techniques of the source publication activity, suitable for real world scenarios.

4.4.2 Comparing Strategies

In the following sections we compare the refresh strategy for scenarios with feed saturation, as presented in Section 4.2 (the $2Steps$ strategy) with four other strategies. All strategies are offline, since they use a priori known information about the average publication frequency λ_i and the divergence value $Div_A(s_i, q, t, T_{r_i})$ for each source s_i.

We briefly present the refresh strategies that we compare with our optimal $2Steps$ strategy presented in Section 4.2:

- *OnlySat* refresh strategy: represents the first step of our optimal $2Steps$ refresh strategy, taken separately. The aggregator refreshes only those b feed sources that have maximum positive window divergence Div_A among the sources that have reached the saturation point.
- *OnlyTau* refresh strategy: represents the second step of our optimal $2Steps$ refresh strategy, considered separately. The aggregator refreshes at time t only those feed sources s_i that have the utility function value $Uti(s_i, q, t, T_{r_i}) \geq \tau$.

- $Uniform$ refresh strategy: refreshes every feed source with the same frequency $b/|S(q)|$ at time cycle t, where b represents the average number of sources that can be refreshed during one time cycle and $|S(q)|$, the total number of feed sources to which the aggregator is subscribed to.
- $Reference$ refresh strategy: refreshes all feed sources at each time cycle t. This refresh strategy may use infinite bandwidth (it is not limited by b), so it is not directly comparable with the others. We chose to present it in order to show the best reference values that can be obtained for the quality measures.

4.4.3 Strategy Effectiveness

In order to verify the effectiveness of our proposed policy we performed experiments comparing our $2Steps$ refresh strategy with the other strategies introduced in Section 4.4.2.

In Figures 4.2 and 4.3, each point represents the feed completeness or the window freshness obtained for different values of b, the average number of sources that the aggregator may refresh during one time cycle, for the different strategies ($2Steps$, $OnlySat$, $OnlyTau$, $Uniform$, $Reference$). The exact results for the circled cases are shown in Table 4.1. The metrics are plotted depending on the total cost, measured as the total number of refresh requests done during 100 cycles simulations.

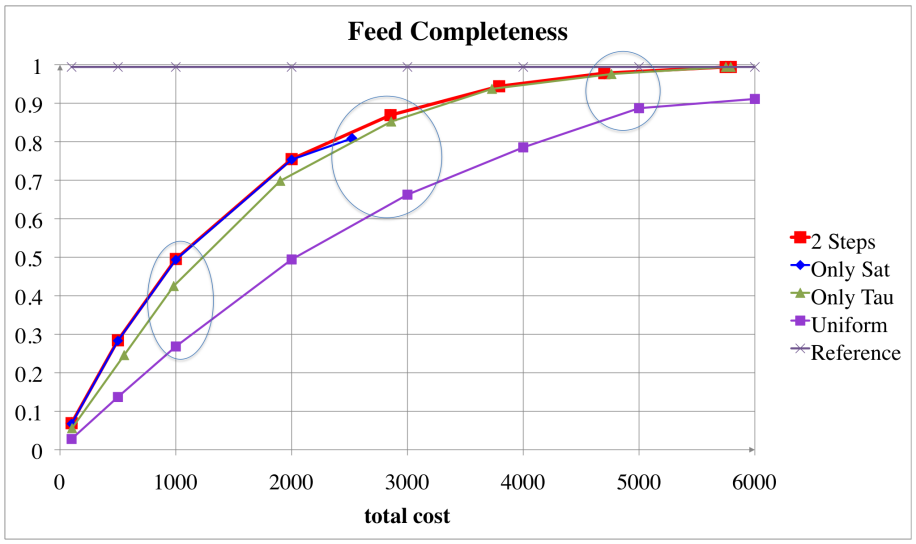


Figure 4.2: Feed completeness

When the aggregator is restricted to refresh few sources (for relatively small values of $b \in (0, 22]$), many sources are not refreshed on time and become saturated very quickly. In this situation, refreshing first the saturated sources (as our optimal $2Steps$ and the $OnlySat$ strategies do) generates significantly better feed completeness values than all the other strategies. We took a sample of this behavior and represented in Table 4.1 the

Figure 4.3: Window freshness

exact measure values circled in Figures 4.2 and 4.3 for an average bandwidth of $b = 10$ refresh requests per cycle. All strategies except *OnlyTau* use the maximal bandwidth that corresponds to a total cost of $b \cdot 100$ cycles $= 1000$ refresh requests.

	$b = 10$			$b = 30$			$b = 50$		
	$\mathcal{C}_F$	$\mathcal{F}_W$	$Cost$	$\mathcal{C}_F$	$\mathcal{F}_W$	$Cost$	$\mathcal{C}_F$	$\mathcal{F}_W$	$Cost$
2Steps	0.4954	0.3671	1000	0.8691	0.7279	2854	0.9781	0.9187	4700
OnlyTau	0.4252	0.4197	982	0.8524	0.7048	2860	0.9759	0.9155	4762
OnlySat	0.4934	0.3671	1000	0.8082	0.6590	2516	0.8091	0.6625	2524
Uniform	0.2682	0.3620	1000	0.6625	0.6398	3000	0.8871	0.7863	5000

Table 4.1: Feed completeness and Window freshness

If the aggregator is allowed to refresh more sources on average (b takes values in $[22, 40]$), the policy behavior changes. This constraint being more permissive, there are fewer sources that reach the saturation point. In this case, refreshing only the saturated sources (the *OnlySat* strategy) obtains poor results. Our $2Steps$ strategy obtains the best results, since it takes advantage of both strategies (*OnlySat* and *OnlyTau*) for refreshing the saturated sources and the sources with the best utility. Compared to the strategy that refreshes based only on the source utility (*OnlyTau* strategy), it obtains better feed completeness and window freshness results for a lower cost. Detailed result values for $b = 30$ that are circled in Figures 4.2 and 4.3 for a total cost of 3000 are presented in Table 4.1.

For large values of the average number of sources that can be refreshed per time cycle ($b > 40$), the optimal $2Steps$ strategy manages to refresh all the sources based on their utility before they reach saturation. In this case, the *OnlySat* strategy performs poorly and the best results for feed completeness and window freshness are achieved by the

strategies that refresh sources based on their utility: the optimal $2Steps$ and $OnlyTau$ strategies. The exact values obtained in the case where $b = 50$ and circled in Figures 4.2 and 4.3 for a total cost of 5000 are presented in Table 4.1.

To sum up, as long as the bandwidth constraints are permissive enough so that the sources do not become saturated before the aggregator gets to refresh them, the $2Steps$ refresh strategy obtains maximum window freshness scores achieved with minimal bandwidth consumption. When the bandwidth constraints are more restrictive and sources get saturated before being refreshed, a refresh strategy has to choose between freshness and completeness. The $2Steps$ refresh strategy focuses on minimizing permanent item loss and obtains maximum feed completeness scores with minimal bandwidth usage.

4.4.4 Strategy Robustness

Using a self adaptive threshold setting algorithm for finding the optimal value of τ in the second step of our $2steps$ refresh strategy makes our policy highly adjustable to the changes of the average source publication frequency λ and of the selectivity $sel(q)$ of the aggregation query q.

Since τ depends on the available bandwidth and the divergence rates of the news feeds, there is no single best value that works well all the time. In order to show the high adaptability of the τ threshold (and consequently, of our optimal $2steps$ refresh strategy robustness), we show a study on the convergence of τ in Figure 4.4.

We plotted the evolution of τ for the optimal $2steps$ refresh strategy when the average number of refresh sources by time cycle is set to $b = 30$, for different τ initial values. From Figure 4.4, we can see that τ always converges to the optimal value. This convergence assures the robustness of our refresh strategy to changes in the publication behavior regarding both frequency and contents of real source feeds.

4.5 Conclusion

In this chapter we have presented a two steps best effort feed refresh strategy that minimizes source divergence and maximizes window freshness and feed completeness achieved with a minimum bandwidth consumption. We started discussing the case of unsaturated feeds and then took into account the feed specific saturation phenomenon. We experimentally evaluated our proposed two steps feed refresh strategy against other refresh strategies and discussed its effectiveness and its robustness.

We continue with introducing in chapter 5 different related works on real RSS feeds characteristics, change models and online estimation methods. In chapter 6 we analyze our own real RSS feeds data set, focusing on the temporal dimension. And we continue in chapter 7 with different change estimation models and associated online estimation methods suited for dynamic RSS feeds.

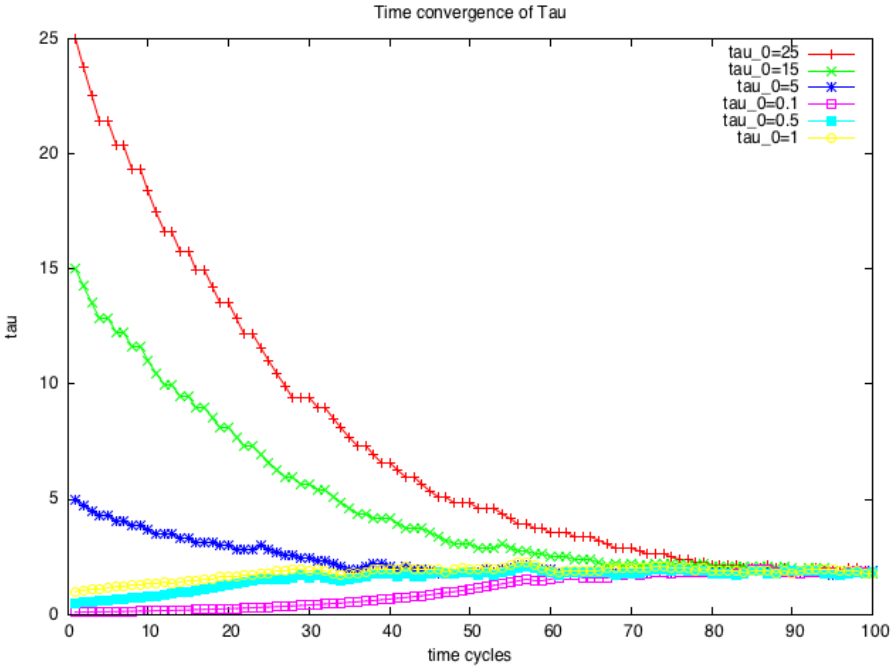


Figure 4.4: Tau convergence

Part III

Data Dynamics and Online Change Estimation

Chapter 5

Related Work

Modern Web 2.0 applications have transformed the Internet into an interactive, dynamic and alive information space. Most of the online web data sources are continually and autonomously changing.For example, online shops update their pages as new products go on sale or change their selling prices. Also, online news magazines update their front page articles periodically whenever there are new developments.

Understanding how web resources evolve in time is important for conceiving tools designed to ease the interaction between people and the dynamic web content. Examples include cache managers that maintain local copies of web data, web crawlers that visit web pages and keep updated the local index, delivery of data from RSS news feeds or the stock market, clients monitoring online auctions or commercial web sites, status change of social network users, collaborative web sites like Wikipedia [wika], etc.

This chapter discusses related work on web data dynamics, with a specific focus on change modeling and estimation. It is organized as follows. First, we briefly present some main issues regarding RSS feed data dynamics and more specifically, publication activity characteristics of real RSS feed data sets. Then, we introduce a survey on existing web change models proposed in the contexts of both web pages and RSS feeds and discuss their underlying parameter estimation methods, with a focus on online estimation.

5.1 Real RSS Feeds Characteristics

Previous efforts in the statistical characterization of RSS resources [HVT+11, RUM+11, LRS05] cover many axes of interest, such as the type, the structure and the size of feeds and items, statistics of vocabulary words occurrences [TPF06, LAT07] and client behavior. In this section, we focus mainly on the RSS feed publication activity analysis, broadly discussed in two recent articles [HVT+11, RUM+11].

The empirical study presented in [HVT+11] relies on a large scale data set acquired over a 8

months period, starting in March 2010 for the Roses French ANR project [rosa]. The data set contains 10,794,285 items originating from 8,155 productive feeds (spanning over 2,930 different hosting sites), harvested from major RSS/Atom directories, portals and search engines, such as syndic8 [syn], Google Reader [good], Feedmil [fee], CompleteRSS [com], etc. Among their many interesting results is the observation that 17% of RSS/Atom feeds produce 97% of the items of the testbed and that $< pubDate >$ RSS element is missing in around 20% of the items. Also, productive feeds (at least 10 items per day) exhibit a more regular behavior with less publication bursts (and are therefore more predictable) than the less productive ones. Feeds are classified into different source types: press (for newspapers and agencies), blogs (for personal blogs), forums (for discussion and mailing lists), sales (for marketing web sites), social media (for micro-blogging such as Twitter [twi], Digg [dig], Yahoo! Groups [yah] and blogosphere) and misc (for news, medical, city/group information, podcasts and others). The publication activity is very different from one source type to another. Feeds from social media are distributed among productive (more than 10 items a day) and slow (less than 1 item a day) classes. Press feeds exhibit a moderate (between 1 and 10 items per day) activity and are more regular, while feeds from forums, sales, blogs and misc are rather slow.

[RUM+11] studies a data set of 200,000 diversified feeds that produced 54 million items, crawled during 4 weeks and collected by feed autodiscovery [rssb] from web pages. Concerning the publication window size, around 25% of the feeds change their window size over time and feeds may have window sizes between zero and several hundred thousands items. A big majority (roughly 27%) of feeds have a publication window size of 10 items, but sizes like 15, 20 and 25 items are also common. They also identified different publication patterns, such as:

- zombie (25% of the total number of feeds) - last publication activity was a long time ago;
- spontaneous (32%) - occasional postings, with no periodic consistency (with an average interval between postings of at least one day);
- sliced (18%) - bursting activity alternated with inactivity periods, classified in between spontaneous and constant patterns;
- constant (4%) - constant publication throughout the entire day (with interval between postings of at most 2 hours);
- chunked (3%) - items are published all at once;
- on-the-fly (0.1%) - feed content is generated on request and all entries have the current time as their publication date.

The rest of the feeds (around 17.9%) include not parsable, unreachable or empty feeds for which the activity patterns can not be defined. The existence of such different feed publication patterns suggests that a refresh strategy that decides when to check the RSS feeds for new updates should take into account their different nature, conclusion that is fundamental for our research direction.

5.2 Change Models

In order to efficiently interact with a dynamic web resource, one must be aware of its update evolution in time. In this section we present different popular approaches used to model the changes of web content, focusing especially on web pages and RSS feeds.

5.2.1 Homogeneous Poisson Process

Experiments reported in the literature [CGM03a, CGM00b] strongly indicate that the *homogeneous Poisson process* is a good model for describing the temporal distribution of updates of web resources, notably for those with average change intervals of at least several weeks. A homogeneous Poisson process is characterized by a constant rate parameter λ and represents a stateless and time-independent random process where events occur with the same probability (rate) at every time point. For estimating λ, this model assumes that the complete change history of each source is known. However, due to bandwidth resource limitations, in reality it is difficult to observe all changes that arise on a web page for building its complete change history. Therefore [CGM03b] proposes methods that estimate the update frequency of a web page based on an incomplete change history. The authors show that a web crawler could achieve 35% improvement in freshness simply by adopting their estimators. Their analysis is based on the hypothesis that the date of the last change or the existence of a change on a web page are known in advance for estimation.

5.2.2 Inhomogeneous Poisson Process

On the other hand, for smaller average change intervals, researchers have shown that the homogeneous Poisson model is no longer suited [BC00, GGLNT04]. The alternative proposed in [SCC07] is to use a *periodic (inhomogeneous) Poisson process* for modeling the publication process of RSS feeds. Their approach is based on experiments that show that most feeds have a highly dynamic but periodic publication behavior, with daily and weekly periodicities.

The same periodic Poisson model is also used in [BGR06]. The authors also propose an *aperiodic Poisson model* by superposing a cyclic component with an acyclic one that adds unpredictable bursts. The use of superposition has several conceptual benefits. First, it can model both a purely cyclic model, by ignoring the burst component and a purely acyclic model indicating no pattern. And furthermore, it separates a situation of temporary bursts from a permanent shift in the change model.

5.3 Online Estimation Methods

Web crawlers for web pages [CGM00a, CGM03a] or RSS feeds [SCC07, SCH+07] are based on a change model estimating for each source its publication activity. These systems usually use *offline* methods, such as average values measured beforehand or learnt during an initial learning phase with access to a complete change history and that are not continuously updated. This assumption is sufficient on short term or for sources with slowly changing and/or low rate publication activities. However, event related feed sources like topic based news feeds or social media feeds (Twitter [twi]) may suddenly change their publication frequency related to a particular event (e.g. twitter hashtag). This data dynamics leads to the necessity of continually updating the publication frequency estimation, using *online* estimation techniques.

5.3.1 Moving Average

[URM+11] proposes an adaptive RSS feed polling strategy based on a set of algorithms that predict a feed's future update behavior. Among the proposed algorithms, they introduce an online estimator, also adopted in [RUM+11], based on the *simple moving average* method that updates continually the predicted interval between two consecutive refreshes. Based on the idea that the recent observations are a good predictor for what happens next, they compute the new update interval as the unweighted mean of the update intervals between the new items fetched at the last refresh moment. If there are no newly published items to fetch, the update interval is increased. The refresh strategy based on the moving average estimation has been experimentally proven to perform better in terms of average delay and bandwidth consumption in comparison with the same refresh strategy based on other offline or fixed estimators.

5.3.2 Exponential Smoothing

Another feed monitoring approach is introduced in [RCYT08]. It dynamically maintains a profile of each feed's content and uses *single exponential smoothing* to learn the change rate of a feed. The learning parameter that controls the tradeoff between the exploitation and exploration effort decreases exponentially in time, using experimentally fixed parameters. They also propose a novelty detection scheme for deciding whether a newly published item is novel by comparing it to the profile of the feed's content. They further integrate the novelty detection into the monitoring strategy by learning the change rate of the novel content of a feed using a similar exponential smoothing approach.

[ZTB+07] propose a different prediction method in the context of information filtering (also referred to as publish/subscribe, continuous querying or information push), where a user posts a subscription (or a continuous query) to the system to receive notifications (using a push protocol) whenever certain events of interest take place (e.g. when a doc-

ument that corresponds to a certain filtering condition becomes available). They further exploit this in MAPS [ZTB+08], an information filtering system for peer-to-peer environments. They compute a node behavior prediction function based ont the likelihood that a node publishes documents relevant to a users subscription based on time series [Cha04] prediction methods. They use simple moving average, but also single and *double exponential smoothing* techniques and validate their methods on different simulated publication behaviors.

5.3.3 Curve Fitting

In the context of web page recrawling, [OP08] introduces refresh strategies based on content-dependent page publication models. They present one offline and two online methods to estimate the page change models. The first online method uses curve-fitting over a generative model and the second one puts conservative bounds to dynamically adjust refresh parameters. Both online change estimation models are based on a data structure called change profile, which is updated at the moment of refresh and which reflects the changes detected that occurred on a web page. The principle of the curve-fitting method is to map a continuous divergence curve obtained from the generative homogeneous Poisson model to the set of points obtained from the change profile data. The principle of the bound-based method is to determine conservative divergence bounds based on the change profile data and use them to adaptively adjust the refresh period. The refresh strategy tested with the two online change estimation methods performs similarly, with a slightly better performance of the curve-fitting method.

5.4 Conclusion

In this chapter we have presented different aspects of web data dynamics. We started with some main issues regarding RSS feed data dynamics, focusing on the temporal evolution of feed publication behavior for different RSS feed sources. We further introduced some change models proposed in the literature both for web pages and RSS feeds evolution. We discussed different online estimation methods that update continually the web change models.

In the following chapters, we present our own analysis on the RSS feeds temporal evolution characteristics that supports our subsequent work. As mentioned in [OP08] and also shown in this chapter, "there is little previous work" on the online refresh strategies topic. We propose different change models adapted for the dynamic RSS feed publication activity and their associated online change estimation methods. We further integrate and test these online estimation models in cohesion with our previously introduced refresh strategy for RSS feeds.

Chapter 6

RSS Feeds Evolution Characteristics

In order to conceive tools that interact with the dynamic web content, it is important to know how the web resources evolve in time. Web data sources are changing autonomously and independently as many of them are generated and managed by users of Web 2.0 applications.

We are interested to know how much new information is published daily by RSS feeds, when exactly it is published and if there are any patterns in the publication activity. In this chapter we propose a general characteristics analysis with a focus on the temporal dimension of real RSS feed sources, using data collected over four weeks from more than 2500 RSS feeds.

We start by describing the way our two RSS feeds data sets are obtained. The feed publication analysis is done from three points of view: publication activity, publication periodicity and publication shapes. First, the publication activity is concerned with the intensity of the source publication process, classifying feeds from very slow to very productive. Second, the publication periodicity searches for daily repetitive publication behaviors of the feed sources. And third, the publication shapes analyzes the daily publication activity forms of the periodic feeds and classifies them in: peaks, uniform and waves.

6.1 Data Set Description

In order to better understand the problem of online change estimation of RSS feeds, we studied two collections of real world RSS feeds focusing on their temporal dimension. Both data sets were crawled using the RoSeS [rosa] feed aggregation system. Further details on its architecture and acquisition module can be found in [rosb]. The two obtained data collections are further used to emulate the publication behavior of real RSS feed sources

and serve us as real world data to experiment on.

Data set 1 represents a selection from a large scale testbed acquired over an eight month campaign in the context of the French ANR project Roses [rosa]. These feeds were harvested from major RSS/Atom directories, portals and search engines, such as syndic8.com [syn], Google Reader [good], feedmil.com [fee], completeRSS.com [com] etc.

We selected only the feeds that publish on at least one of the main categories showed in Figure 6.1.

Figure 6.1: Feed categories

Each category is defined by a list of keywords. For example, category "economy" is defined by the following list of keywords: economy, business, trade, transaction, deal, operation, commercial, finance, market, sale, auction, euro, money, budget, crisis. We consider that a feed publishes on a certain category if at least one of the following conditions holds:

- it has at least one keyword specific to that category in its feed categories
- it has published at least one item that contains at least one keyword specific to that category in its item categories
- it contains at least one keyword specific to that category in the url of the feed.

We obtained 1969 RSS feeds, out of which we selected only 1658 that published at least one item during the four week crawling period from 14 March to 10 April 2011.

Data set 2 was acquired from a manually chosen list of RSS news feeds of different online newspaper websites, both French (such as Le Monde, Le Figaro, AFP) and international (such as CNN, New York Times, Euro News). We obtained 1339 RSS feeds, out of which we selected only 963 that published at least one item during the four week crawling period from 14 March to 10 April 2011.

6.2 Publication Activity

From the data sets described in the previous section, we collected statistics on the daily publication frequency of the various feeds.

We group in the same *activity class* the feeds with similar publication frequency. In Figure 6.2 we show the distribution of feeds for various activity classes for the two different data sets. For data set 1 (random feeds harvested from major RSS/Atom directories and portals), the distribution shows that feeds with very slow publication activity (less than 1 item per day) are predominant and represent approximate 50%. Feeds that publish on average between 1 and 10 items daily represent 37%. Feeds from data set 2 (news feeds harvested from French and international online newspaper) are globally more productive: there are less feeds with very slow publication activity (approximative 25%), while feeds with moderate publication activity (between 1 and 10 items daily) represent the majority (52%). For both data sets, feeds that are very productive, publishing more than 10 items per day represent roughly 20%: 14% in case of data set 1 and 23% for the news feeds (data set 2).

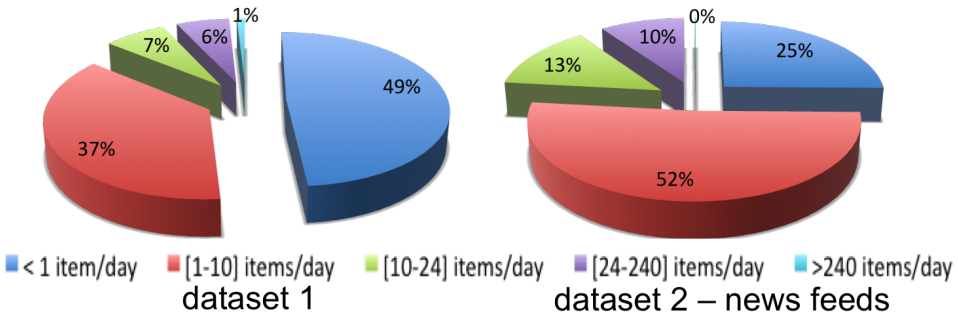


Figure 6.2: Feeds per activity class

Whereas the number of productive feeds is quite small, it has been shown in [HVT+11] that they are the ones that produce most of the published items: 17% of RSS/Atom feeds produce 97% of the items.

6.3 Publication Periodicity

It is widely accepted that the past change represents a good predictor of future change. This works well especially for those types of feeds that have a foreseeable publication activity, for example, feeds that have a daily periodicity, i.e. publish daily the same number of items at the same hours. In this sense, measurements on real data done in [SCC07] show that most of the daily posting rates of feed sources are stable, at least for their data set, within the three month period they used for their experiments. But there are also feeds whose publication behavior vary in time, both in the number of daily

published items and in the shape of publication activity.

In order to detect changes in publication frequency, for each hour (time slot $i \in [0, ...23]$) of a day j, we logged the number of items x_{ij} published by a feed and then computed the *mean* $\mu_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij}$ and the *standard deviation* $\sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_{ij} - \mu_i)^2}$ for all the $n = 28$ days.

As an example, in Figures 6.3 and 6.4 we represented the average (pink bars) and the standard deviation (vertical lines) of the number of published items at different time slots, one for each hour of the day, for a periodic and an aperiodic feed.

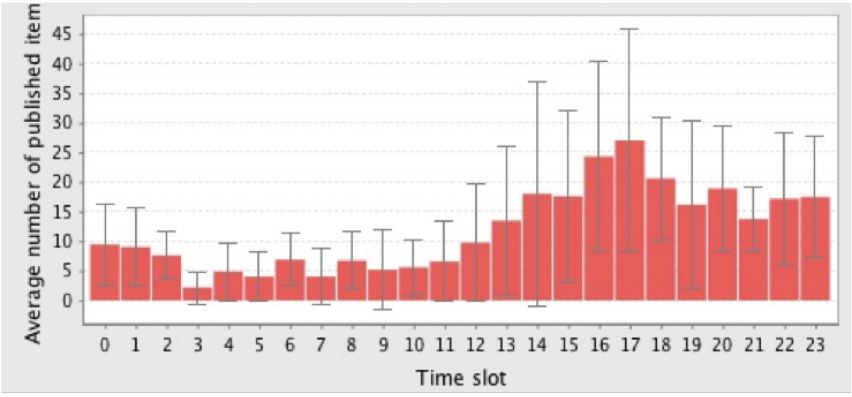


Figure 6.3: Periodic publication behavior

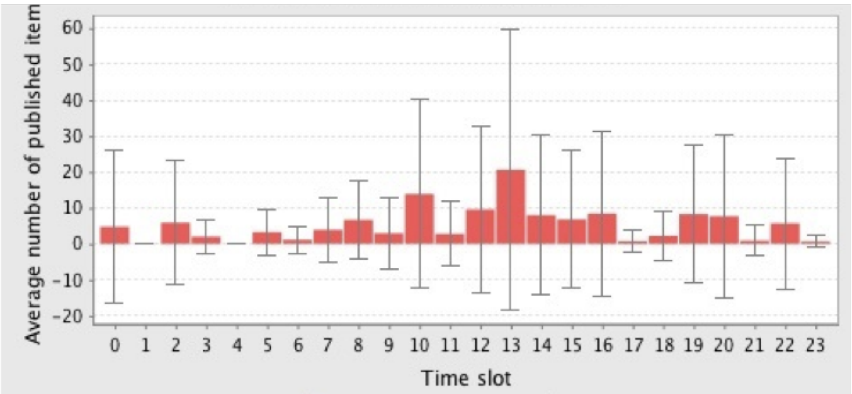


Figure 6.4: Aperiodic publication behavior

We consider that a small *coefficient of variation* CV value is representative for feed sources with high periodicity.

$$CV = \frac{1}{24}\sum_{i=0}^{23}\frac{\sigma_i}{\mu_i} \qquad \text{where } \mu_i \neq 0$$

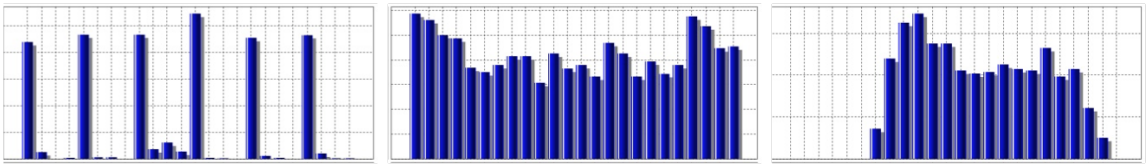
When the mean values are close to zero, the coefficient of variation becomes sensitive to small changes in the means (approaches infinity) and this measure becomes inappropriate

for testing sources with a low publication activity. We found out that 20% of the sources that publish more than 10 items per day and 50% of the ones that publish more that 48 items per day have an average standard deviation to mean ratio smaller than 1 ($CV \leq 1$).

6.4 Publication Shapes

We also studied the feed collection looking for different "shapes" in the daily publication activity. The shape of a daily publication model highly depends on the data processing and information generation process "behind" each feed. Some feeds may be generated by human activity, while others may be based on some automatic publication process.

In Figure 6.5 we represented with vertical blue bars the average number of items published at different hours of the day for different periodic source feeds. We classified the feeds in three different categories, as shown in Figure 6.5: feeds that have *peaks*, usually generated by an automatic publication robot, that have a *uniform* publication activity, such as in the case of a news aggregator that continuously gathers news published around the world and those that exhibit *waves*, following the regular daily schedule of a human activity.



(a) Peaks (b) Uniform (c) Waves

Figure 6.5: Publication shapes: peaks, uniform and waves

Shape Discovery Heuristic. This classification has been obtained by using a shape discovery heuristic that automatically groups different publication activity shapes into one of the three categories presented above: peaks, uniform or waves.

Let the publication behavior of a periodic feed source be described by different publication frequency values λ_i that correspond to different time slots i (we consider that there are N time slots; e.g. if a time slot corresponds to one hour, then $N = 24$). Given the feed source publication behavior, let $\lambda_{avg} = \frac{1}{N}\sum_{i=1}^{N}\lambda_i$ represent the average number of items published by the source during an hour time slot. We define two thresholds, $th+$ and $th-$ as:

$$\begin{aligned} th^+ &= \epsilon_1 \cdot \lambda_{avg} \qquad \text{where } \epsilon_1 > 1 \\ th^- &= \epsilon_2 \cdot \lambda_{avg} \qquad \text{where } \epsilon_2 < 1 \end{aligned}$$

The intuitive idea is to fix the values of the two thresholds such that the time slots i that have the publication rate λ_i less than the inferior threshold th^- to be considered as having

negligible / insignificant publication activity, those that exceed the superior threshold th^+ to be considered as having *very high activity* and those that have values in between the two thresholds as having an *average* activity.

The shape discovery heuristic presented in the Algorithm 6.1 starts by counting the number of time slots i for which the publication rate λ_i is above the superior threshold th^+ (counter $\#th^+$) and those for which it is below the inferior threshold $th-$ (counter $\#th^-$). Next, we consider that a feed source publishes with peaks if it has all its publication rates λ_i outside the interval $[th^-, th^+)$, alternating very high with insignificant publication activity. As an alternative, we can consider less restrictive conditions and allow a feed to be classified as publishing with peaks if it has a big majority (e.g. 80%) of time slots i that have λ_i outside the interval $[th^-, th^+)$. Similarly, we label all source feeds with publication rates λ_i for all time slots i (or for a big majority, e.g. 80%) inside the interval $[th^-, th^+)$ as having a uniform publication shape. All the other feeds re considered to have a publication activity shape with waves.

Algorithm 6.1 Shape Discovery Heuristic

Input: PEAKS, UNIFORM, WAVES = sets of feeds with specific publication shapes
λ_i = publication frequency during time slot i of feed s
λ_{avg} = average publication frequency during a time slot
$\epsilon_1 > 1$, $\epsilon_2 < 1$
$th^+ = \epsilon_1 \cdot \lambda_{avg}$, $th^- = \epsilon_2 \cdot \lambda_{avg}$
$\#th^+ = 0$, $\#th^- = 0$
for $i = 1$ **to** N **do**
  if $\lambda_i \geq th+$ **then**
    $\#th^+ = \#th^+ + 1$ // count the number of time slots i with very high publication activity
  end if
  if $\lambda_i < th-$ **then**
    $\#th^- = \#th^- + 1$ // count the number of time slots i with insignificant publication activity
  end if
end for
if $\#th^+ + \#th^- == N$ **then**
  PEAKS.add(feed s) // feed s alternates between very high and insignificant publication activity
else if $\#th^+ == 0$ and $\#th^- == 0$ **then**
  UNIFORM.add(feed s) // feed s has only average publication activity
else
  WAVES.add(feed s)
end if

In Figures 6.6 and 6.7 we show the distribution of feeds for various activity classes and publication shapes for data sets 1 and 2 (news feeds). The distributions are similar for

the two data sets and show that feeds with very slow publication activity (columns on the left side) tend to publish more with peaks, the uniform pattern is very much present in feeds with very high publication activity (columns on the right side) while the wave shape appears in feeds with low, medium and high publication frequencies.

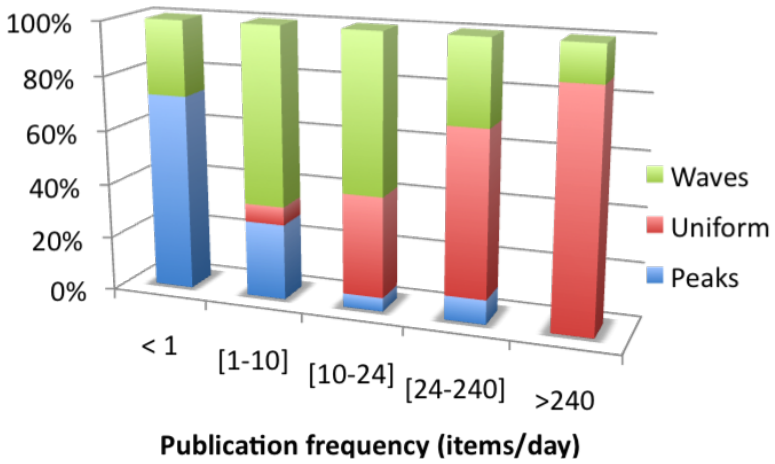


Figure 6.6: Publication shapes per activity class - data set 1

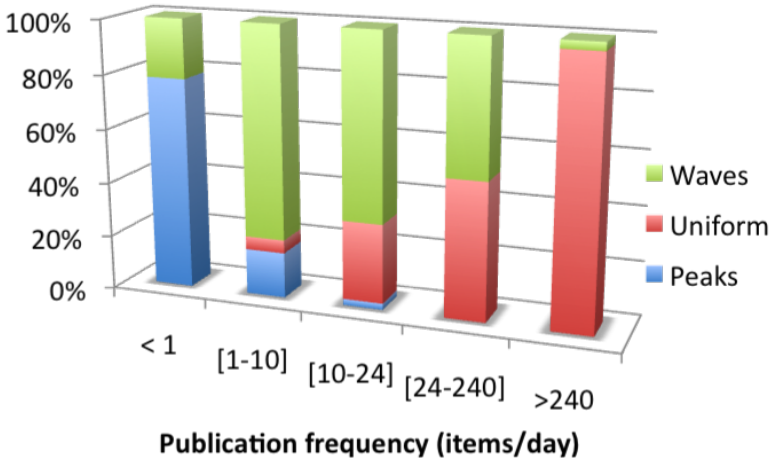


Figure 6.7: Publication shapes per activity class - data set 2 news feeds

6.5 Conclusion

In this chapter we have presented a general characteristics analysis with a focus on the temporal dimension of real RSS feed sources, using data collected over four weeks from more than 2500 RSS feeds. We analyzed it from three points of view: publication activity, publication periodicity and publication shapes. We started by looking into the intensity

of the feed publication activity and into their daily periodicity features. We also studied the shapes of their publication activity and discovered three characteristics shapes: peaks, uniform and waves.

Inspired by the observations made on the real RSS feeds publication behavior, in the next chapter we propose two change estimation models that reflect different types of feed publication activities. Furthermore, we propose two corresponding online estimation methods suited for dynamic RSS feeds that are continually updated in order to reflect the changes in the real feed publication activities.

Chapter 7

Online Change Estimation Techniques

RSS feeds represent a typical example of highly dynamic web content that is updated regularly. As shown in chapter 6, the type of changes are very different from one feed to another. Some RSS feeds publish several times during one hour and others are updated less than once a day. A good source publication model is very important to efficiently predict when exactly the source publishes new items and therefore decide when is the optimal moment to refresh it.

In this chapter we focus on the problem of estimating the change frequency of dynamic RSS feeds. Our goal is to improve the refresh strategies of RSS aggregators, but other web data processing systems like web crawlers or web data warehouses may as well benefit from the techniques presented in this chapter. For that, we propose two online estimation methods that correspond to different RSS publication activity models.

We start by introducing the general problem of online change estimation and emphasize its importance. We present the single variable and the periodic publication models and propose methods for estimating them online. We also detail a hybrid publication model and discuss a dynamic algorithm for adjusting the value of the estimation smoothing parameter. Experimental evaluation is done on real world RSS feeds. We analyze the cohesion of our online estimation methods with different refresh strategies and examine their effectiveness on feed sources with different publication behaviors.

7.1 Online Change Estimation

Large scale web applications like web search engines, web archives, web data warehouses, publish-subscribe systems and news aggregators have to collect information from a large number of dynamic web resources. In order to accomplish this task efficiently, these

systems depend on appropriate source *publication models* for deciding when to refresh each source in order to maximize one or several quality criteria under limited resources. Content independent models [CGM03b] estimate the probability that a source has changed at least n times at some time instant t, whereas content dependent models [OP08] might include some heuristics for estimating the importance of change between two versions.

We consider the case of an RSS aggregator node which is subscribed to a collection of sources. Let T_r represent the last time moment when source s has been refreshed by the aggregator. We remind the definition of the (stream) divergence function $Div(s, t, T_r)$ introduced in 2.1.1 as the total number of new items published by the source s in the time period $(T_r, t]$ that were not yet fetched by the aggregator. Obviously the quality (preciseness) of the divergence estimation is important for the quality of the corresponding refresh strategy [CGM00a, CGM03a].

A traditional way for estimating divergence is to use the publication behavior of the source s characterized by the time dependent *publication frequency* variable $\lambda(s, t)$, defined as the average number of items published by source s in some give time unit (e.g. second, minute, etc.). Divergence can then be defined as an integral of publication frequency $\lambda(s, t)$ over time:

$$Div(s, t, T_r) = \int_{T_r}^{t} \lambda(s, x) \cdot dx \tag{7.1}$$

In practice, refresh strategies use a discrete time dimension, where time periods are divided into time units of fixed size and divergence is defined by a sum of divergence estimations for the intervals (see Section 7.2).

7.1.1 Online and Offline Change Estimation

The general refreshing process illustrated in Figure 7.1 is accomplished by (1) the *refresh strategy* which *uses* the publication model for estimating the divergence and the next refreshing time moment of each source and (2) the *change estimator* which generates and updates the publication model. In an *offline scenario* we ignore the presence of the change estimator module. The refresh strategy uses a precomputed publication model and its update is a problem treated separately. *Online estimation* interleaves both tasks and each new observation (obtained by a refresh) is used immediately for updating the publication model.

7.1.2 Importance of the Online Change Estimation

Keeping the estimated publication frequency of a source constant over a long period of time can represent an important source of errors if the source publication activity changes in time.

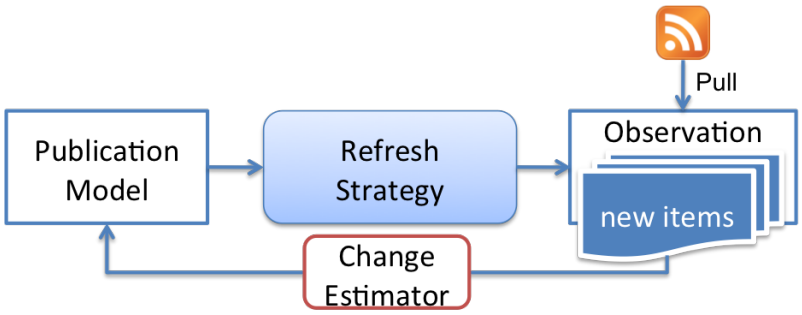


Figure 7.1: Online estimation

This is illustrated in Figure 7.2 showing the evolution of the real and estimated divergence functions of a source during a day. Figure 7.2 compares (for a given source) the real divergence values (red curve) with the estimated values using a constant publication frequency (offline estimated divergence as the green curve) and the estimated values using an adaptive publication frequency (online estimated divergence as the blue curve). In both curves, the source is refreshed at regular time intervals which resets the divergence values to 0.

The green estimated divergence function presented in Figure 7.2 increases with a constant slope because it is based on a constant publication frequency (previously learnt in an offline manner and not updated afterwards). Differently from this case, the blue estimated divergence function in Figure 7.2 is computed based on a publication frequency that continuously adapts its value in time (online estimation), converging to a zero publication frequency when the source does not publish anything and increasing as the source starts publishing. The estimation is obviously better on average in the second case, when the online estimated divergence curve is the closest to the real one.

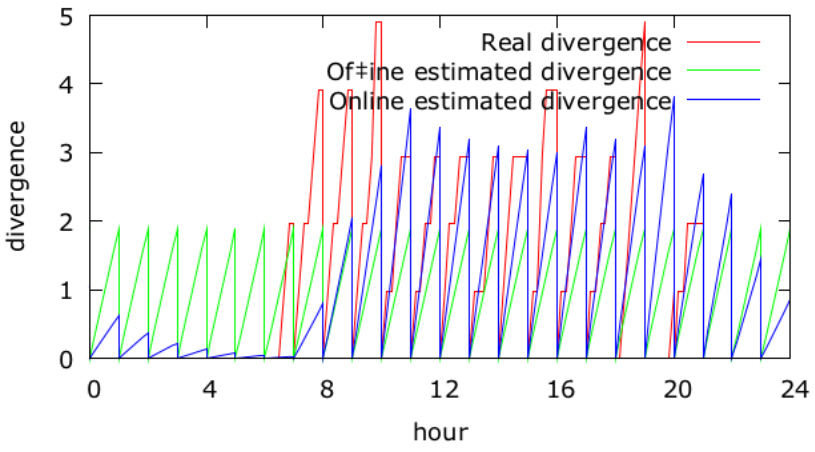


Figure 7.2: Real vs. estimated divergence

7.2 Online Change Estimation for RSS Feeds

Our approach for estimating the change rate of RSS feeds is strongly inspired from standard results in time series analysis [Cha04]. These techniques are used to predict future time series values based on past observations and are usually based on the hypothesis that both observations and predictions are done at constant time intervals. In our case, observations are made at the refresh moments decided by the refresh strategy. This makes the prediction process less precise than in the case of classical time series model usage.

We base our online estimation methods on observations of the number of occurred changes, i.e. new items published by a feed. In the particular case of working with RSS feeds, we could have chosen to use the specific RSS field $< pubDate >$ in order to find out exactly the publication date of each item. Nevertheless, we prefer to ignore this attribute for two reasons: first, because not all feed items offer this information ([HVT+11] reports that $< pubDate >$ is missing in about 20% of items) and second, in order to maintain the generality of the estimation methods, that allows them to be used in other different contexts (e.g. web pages).

We introduce next three different publication models for RSS feeds and their corresponding online change estimation techniques.

7.2.1 Single Variable Publication Model

Divergence Estimation

Our first publication model represents the publication frequency of a source s at time t by a single variable, $\lambda(s,t)$. Let T_r represent the time instant of the r^{th} refresh of s and $\lambda^r = \lambda(s,T_r)$ be the change rate of source s estimated at time instant T_r. Then the divergence of s at time instant $t \in [T_r, T_{r+1})$ can be simply estimated by the following formula:

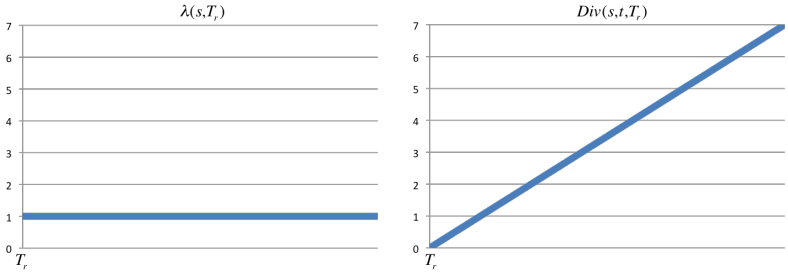
$$Div^{est}(s,t,T_r) = (t - T_r) \cdot \lambda^r$$

The form of the single variable publication frequency and of the divergence function are presented in Figure 7.3.

Publication Frequency Update

Let $x^{r+1} = Div(s, T_{r+1}, T_r)$ be the number of new items published since the last refresh at T_r and observed at T_{r+1}. The newly estimated value of the publication frequency is obtained by single exponentially smoothing the new observation with the previous estimation:

$$\lambda^{r+1} = \alpha \cdot \frac{x^{r+1}}{(T_{r+1} - T_r)} + (1 - \alpha) \cdot \lambda^r \tag{7.2}$$



(a) Publication frequency (b) Divergence function

Figure 7.3: Single variable publication model

This estimation method relies on all previous observations, with exponentially decaying weights, parameter $\alpha \in [0, 1]$ representing the smoothing constant.

7.2.2 Periodic Publication Model

Our second estimation model of publication is based on the hypothesis of periodicity. In this case, the publication frequency of a source is described as a periodic function with some (constant) period Δ_T: $\lambda(s, t) = \lambda(s, t + \Delta_T)$.

We use a *discrete* representation of the publication frequency as a table $P(s)$ of n values, each corresponding to a time slot $[t_i, t_{i+1})$, $i \in \{0, ...n-1\}$. Each time slot is of constant size $t_{i+1} - t_i = \Delta_T/n$. We will call $P(s)$ the publication model of s. Then $\lambda_i(s, t)$ corresponds to the $(i + 1)^{th}$ value in $P(s)$ where $(t \bmod \Delta_T) \in [t_i, t_{i+1})$ (i is the time slot covering t). In the following we denote by λ_i the average publication rate of source s during time slot i. In our experiments (Section 7.3) we use a daily publication model where $\Delta_T = 24$ hours, $n = 24$ time slots of 1 hour each.

Divergence Estimation

Let T_r represent the time instant of the r^{th} refresh of s and $P^r(s) = \{\lambda_i^r\}$, $i \in \{0, ...n-1\}$ be the publication model of s estimated at time instant T_r. Then the expected divergence of s at time instant $t \in [T_r, T_{r+1})$ can be estimated by the following formula where i corresponds to the time slot containing T_r and there are $k + 1 = (\lceil t \rceil - \lfloor T_r \rfloor) \cdot n/\Delta_T$ time slots "covered" by the interval $[T_r, t)$ (the definitions are illustrated in Figure 7.4):

$$Div^{est}(s, t, T_r) = \int_{T_r}^{t} \lambda_i^r(s, x) \cdot dx =$$

$$= \lambda_i^r(t_{i+1} - T_r) + \Delta_T/n \cdot \sum_{j=i+1}^{i+k-1} \lambda_{(j \bmod n)}^r + \lambda_{i+k}^r(t - t_{i+k})$$

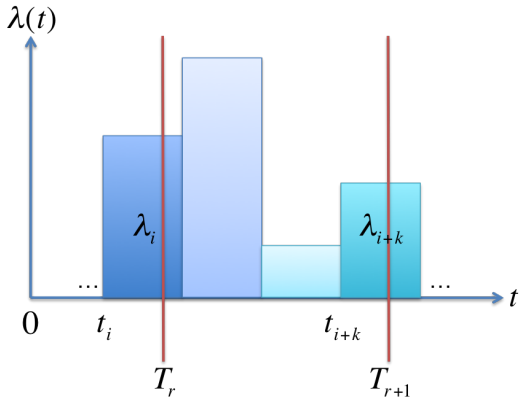
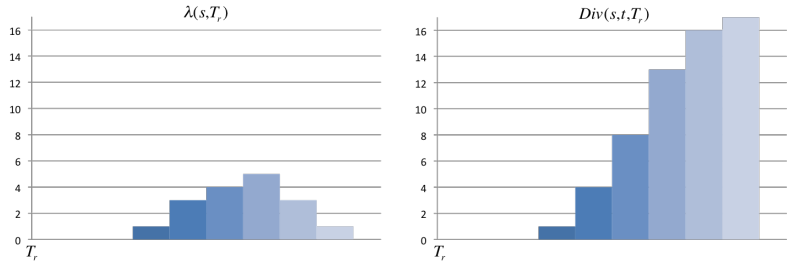


Figure 7.4: Periodic publication model - Publication frequency

The form of a periodic publication frequency and of the divergence function are presented in Figure 7.5.



(a) Publication frequency (b) Divergence function

Figure 7.5: Periodic publication model

Publication Frequency Update

Suppose that the aggregator refreshed some source s at some time moments T_r and T_{r+1} that correspond to time slots i and $i+k$. At T_{r+1}, the aggregator fetches $x^{r+1} = Div(s, T_{r+1}, T_r)$ new items published since the last refresh at T_r. The intuitive idea of the model update is to distribute the last observed items x^{r+1} in the time interval $[T_r, T_{r+1})$. This distribution is done *proportionally* to the expected divergence $Div^{est}_{r+1} = Div^{est}(s, T_{r+1}, T_r)$ estimated using the values of λ^r_j that correspond to the time interval $[T_r, T_{r+1})$. We compute λ^{r+1}_j as the newly predicted value of λ_j that corresponds to time slot j as follows:

$$\lambda_j^{r+1} = \begin{cases} \alpha \cdot \frac{\lambda^r_j}{Div^{est}_{r+1}} \cdot x^{r+1} + (1-\alpha) \cdot \lambda^r_j & \textbf{if } j \in \{i, ...i+k\} \\ \lambda^r_j & \textbf{otherwise} \end{cases} \tag{7.3}$$

where $\alpha \in [0, 1]$ represents a smoothing parameter that is used to give more or less weight to recent observations. This reestimation formula corresponds to a maximum likelihood estimate of the publication frequencies λ_j based on the observation x^{r+1} at iteration $r+1$, smoothed with the estimates at previous iteration r.

7.2.3 Hybrid Publication Model

The choice of which publication model to use for refreshing a certain source depends on several factors, such as the type of the source publication activity. Moreover, using one publication model to refresh a source may generate the best results, but only for a limited period of time. A change in the source publication behavior may trigger the need to change the publication model used by the refresh strategy to determine the next moment that source should be refreshed.

In order to avoid such decisions that demand knowledge that is not always available, one option is to use a *hybrid publication model* that dynamically switches between the single variable 7.2.1 and the periodic 7.2.2 publication models. The principle of the hybrid solution is composed by two steps. First, if at time moment T_r a refresh is done, the real divergence value x^r is found and both publication models are updated. The real divergence value can be compared with the different divergence values estimated for the two publication models, computing a divergence error function (as the one proposed in Equation 7.4). Second, in order to determine the next refresh time moment T_{r+1} of a source, the refresh strategy takes this decision based on the publication model that had a minimum divergence error computed at the time T_r of the last refresh.

7.2.4 Smoothing Parameter Adjustment

The smoothing parameter α that appears in the frequency update Equations 7.2 and 7.3 represents a constant taking values in $[0, 1]$, used to give more or less weight to the most recent observations to the detriment of the older ones. The actual value of α depends on the type of the source, on the refresh frequency and on how close close to convergence is the the source publication model. The weighting factors for each older data point decrease exponentially, never reaching zero. The speed at which older observations are dampened (smoothed) is a function of α. When α is close to 1, dampening is quick and when α is close to 0, dampening is slow. The value of α must be chosen such that the errors between the estimated values and the observed ones (in our case, the divergence errors) should be minimized.

Intuitively speaking, if the estimated publication model of a source is close to the "real" one, the divergence errors values that are obtained should be small and, the same way, when the estimated publication model has diverged from the real one, big divergence errors are expected.

Dynamical α Adjustment Heuristics. We propose a heuristic for dynamically adjusting the value of the smoothing parameter α that moves its value in the direction of the divergence errors minimization.

We define first a *stability indicator* $= n_1/n_2$ as the ratio of the average of the most recent n_1 divergence errors to the average of the last n_2 divergence errors, where $n_1 < n_2$. The numerator reflects the errors recorded in the near past and the denominator represents the errors observed during a sliding history window. For example, if $n_1 = 3$ and $n_2 = 10$, then the stability ratio compares the most recent 3 errors with the average of errors observed during the last 10 observations.

We consider the publication profile of a source as being *stable* if the stability ratio remains approximately constant (e.g. stability ratio ≈ 1 or ≤ 1). If the publication profile is stable, then it is converging and the smoothing parameter α can be decreased (e.g. $\alpha := \alpha \cdot \gamma_1$, where $\gamma_1 < 1$). Similarly, we consider that the source publication profile is *unstable* if the last observed error values are increasing or in other words, if the stability ratio is bigger than a threshold (e.g. stability ratio > 1). In case the publication profile is unstable, then it is diverging and the smoothing parameter α can be increased (e.g. $\alpha := \alpha \cdot \gamma_2$, where $\gamma_2 > 1$). Furthermore, imposing some superior and inferior bounds for α restrains it from diverging.

7.3 Experimental Evaluation

In this section, we compare the online estimation techniques that correspond to the single value and the periodic source publication models with respect to the three classes of feed publication shapes, e.g. peaks, uniform, waves. We evaluate their performances in cohesion with different refresh strategies based on real RSS feeds data collected during a four week period, introduced in chapter 6.

7.3.1 Experimental Setup

We focused our interest on feeds with a high publication activity and we selected for our experiments three subsets of 10 feed sources each, representative for the three publication shapes, having a publishing activity of at least 10 items per day.

We emulated the source publication activity by constructing a cycle based environment, where a cycle corresponds to a time unit of duration 10 minutes. Furthermore, we worked with a normalized source publication model, i.e. we did not consider anymore that a source published x items during a certain time slot, but that it published x/N, where N represents the total number of items published by the source during an entire day. Working this way, we focused ourselves on estimating the shape of a source publication activity and we avoided the influence of any strong fluctuation in terms of total number of items published daily.

Choosing the optimal value of the smoothing parameter α depends on the type of the source, on the refresh frequency and on the level of convergence of the source publication model. In each case, we chose an experimentally found value of α such that it minimizes the divergence errors, usually using values in the $[0.01 - 0.2]$ interval.

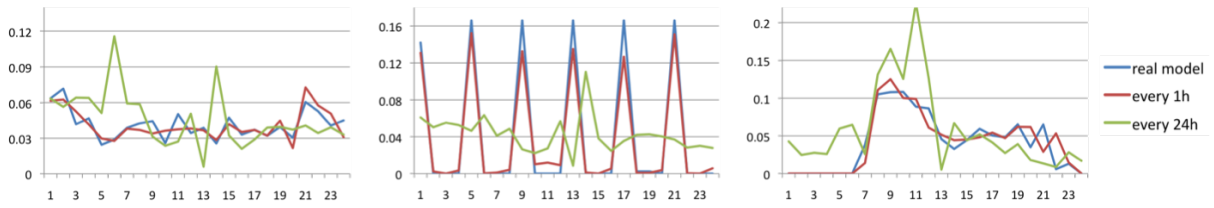
7.3.2 Online Estimation Evaluation

In order to evaluate the different techniques for estimating online the publication frequency of a source, we used a *uniform random* refresh strategy (refreshes are done at irregular intervals of time that are uniformly distributed around a fixed average value). For example, a source that is refreshed on average every 1 hour means that it can be refreshed with the same probability at any time interval between 10 minutes and 2 hours. We put all sources in the same initial conditions, initializing their publication frequencies at 0 and starting the evaluation after an initial warm up period.

Robustness of the Periodic Publication Estimation

We tested the robustness of our periodic publication estimation, how it acts to sudden changes in the publication behavior of the sources and how it is influenced by the refresh frequency used by the strategy. For that, we created an artificial source by concatenating publication activities from three sources with different publication shapes: 16 weeks of uniform, followed by 16 weeks of peaks and followed by 16 weeks of waves.

Experiments were done using a *uniform random* strategy that refreshed the source every 1 hour and every 24 hours on average. We registered the estimated daily publication models at the end of each week. We also computed a reference daily publication model as an average done on the 7 days of source publication activity previous to the measurement moment: λ_i is the average number of items published by a source in time slot i during the previous 7 days. In Figure 7.6 we compare the reference model to the online estimated daily publication models of the artificial source just before each change in the publication behavior, i.e. at the end of 16th, 32nd and 48th week (time moments circled and marked with vertical blue lines in Figure 7.7).



(a) After 16 weeks: Uniform (b) After 32 weeks: Peaks (c) After 48 weeks: Waves

Figure 7.6: Daily publication model: real vs. estimated model

Furthermore, we show the 24-dimensional Euclidean (2-norm) distance between the real and the estimated daily publication models after each week in Figure 7.7.

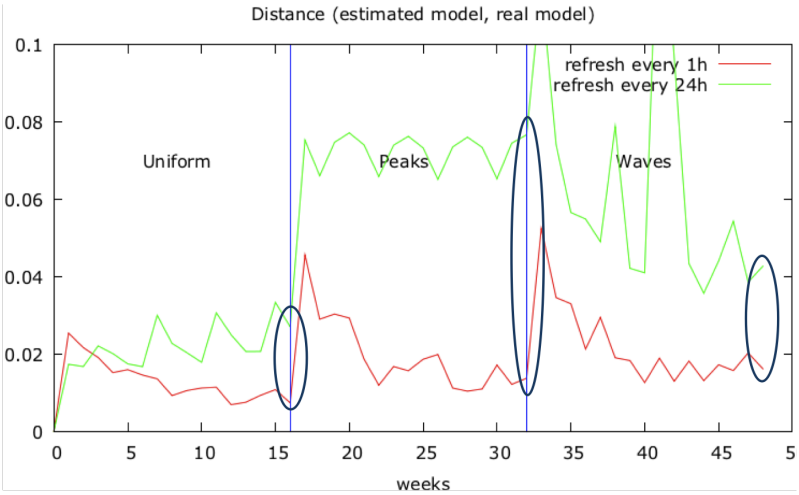


Figure 7.7: Distance between real and estimated periodic model

Figures 7.6 and 7.7 prove the bad influence of a small refresh frequency can have on the quality of the estimation process. Convergence speed of the publication estimations are shown in Figure 7.7: while the estimated daily publication model obtained with an average refresh frequency of 1 hour converges rapidly towards the referenc model, the estimated model obtained with a refresh done every 24 hours oscillates and diverges in time.

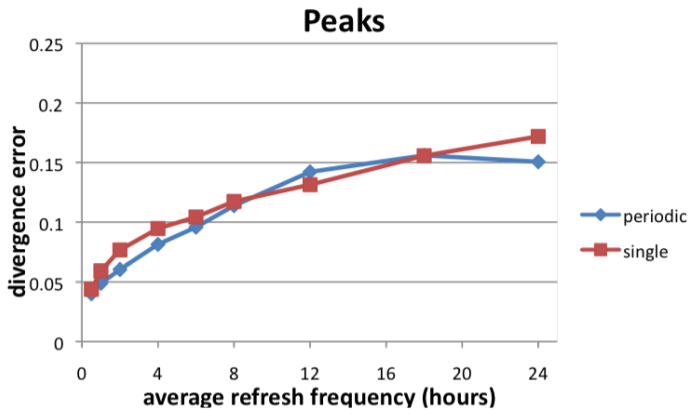
Online Estimation Quality

At each cycle t, we computed the root mean squared error of the estimated divergence (defined in Section 7.1) for all sources $s_i \in S$, separately for the periodic and for the single variable publication model:

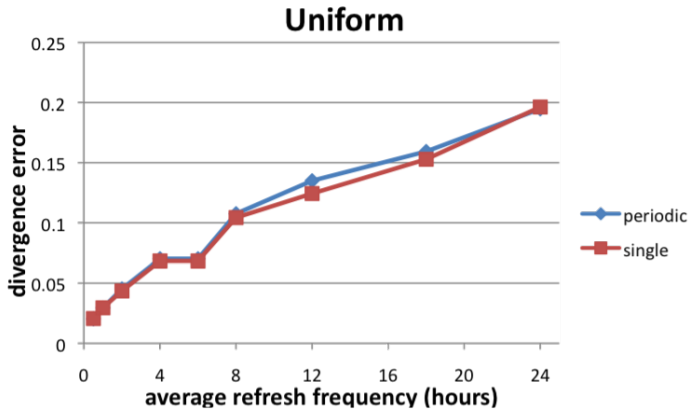
$$divErr = \sqrt{\frac{1}{|S|} \cdot \sum_{s_i \in S} (Div(s_i, t, T_r)^{real} - Div(s_i, t, T_r)^{est})^2} \tag{7.4}$$

Results are presented in Figure 7.8, separately for the three types of sources with different publication shapes: peaks, uniform and waves. Each point represents the average of the root mean squared divergence errors computed during the simulation, that were obtained for different refresh frequencies. The values used for the refresh frequencies are shown in hours and they range from a refresh done every 30 minutes to every 24 hours on average.

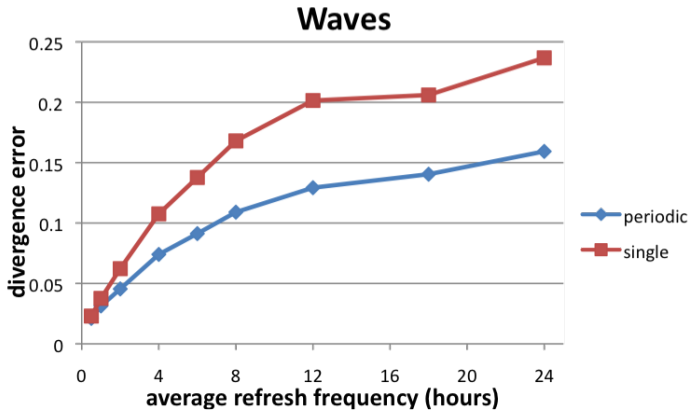
Experiments show clearly that in the case of waves, the periodic estimation obtains better results than the single variable one in terms of minimal divergence error. Since it is more precise, it estimates better the wavy source publication behavior, no matter how often the



(a) Peaks



(b) Uniform



(c) Waves

Figure 7.8: Divergence error

sources are refreshed and thus, how often the publication model is updated. In the case of peaks, the difference between the two publication estimations is less striking. When the sources are refreshed often and therefore the learnt periodic publication model is precise, the periodic estimation obtains smaller divergence errors. As the sources are refreshed less frequently, the single variable estimation becomes as good as the periodic one; this happens for two reasons: first, the periodic model becomes less accurate and thus it diminishes its performance and second, our feed sources exhibit their peaks at very regular intervals, e.g. every 4 hours, as shown in Figure 6.5a. This favors the single variable publication model when the refresh frequency is larger than the average interval in between peaks, i.e. 4 hours. As for the uniform sources, both single and periodic publication estimations perform similarly, with the observation that the single variable publication model should be preferred because it is much more simple to use and update. Uniform feeds represent 57% of the feeds with high publication rate (more than 1 item published per hour), as we observed on our real feeds data sets (chapter 6).

7.3.3 Integration of Online Estimation with 2Steps Refresh Strategy

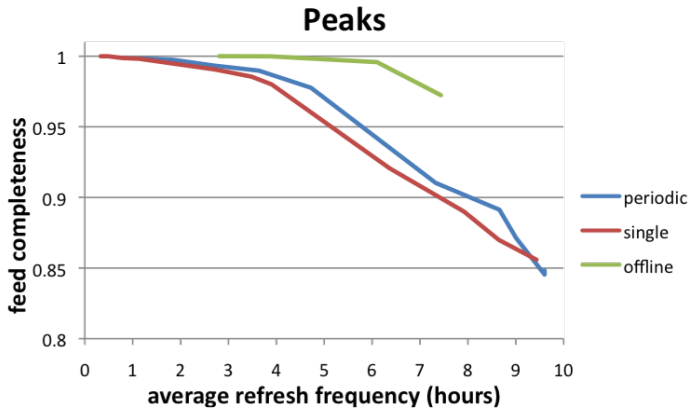
We also integrated and tested our online estimation techniques with the optimal $2Steps$ refresh strategy introduced in Section 4.2, whose efficient results highly depend on the quality of the used publication models.

It is important to mention that we ignored the saturation problem when updating the publication models. We chose to do that in order to help the online estimation by giving it unbiased information as input.

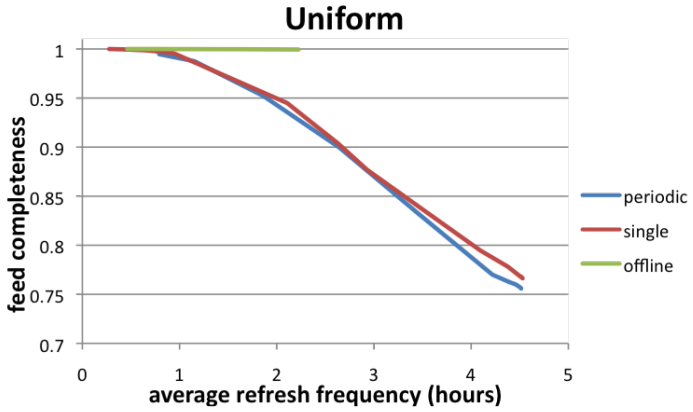
As before, we evaluated the online estimation quality by measuring the divergence error (Equation 7.4). The results obtained for the sources with different publication shapes are similar to those obtained with the uniform random refresh strategy (see Figure 7.8).

Furthermore, we tested the effectiveness of the $2Steps$ refresh strategy in terms of feed completeness and window freshness (quality measures defined in chapter 2) and present them in Figures 7.9 and 7.10. The green curves represent the case when the $2Steps$ refresh strategy uses a priori know information about the source publication models, while the red and the blue curves reflect the results obtained by the $2Steps$ refresh strategy when using online estimation techniques for the single variable and the periodic source publication models.

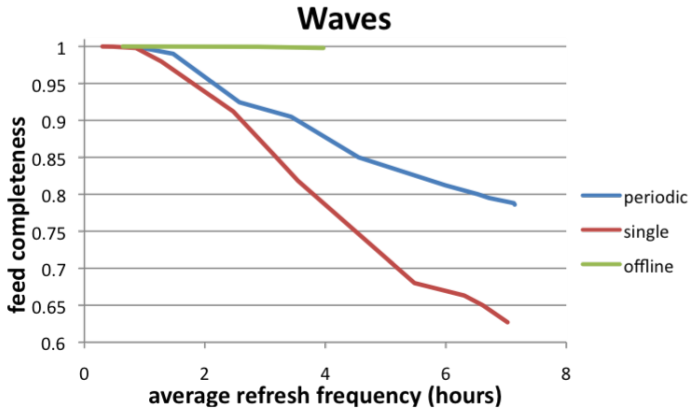
When sources are refreshed very frequently (big bandwidth), both periodic and single variable publication estimation give very good results in terms of feed completeness and window freshness, independently of the source publication shapes. Frequent refreshes alone assure high scores for quality measures and besides that, good convergence for both periodic and single variable publication models. In the case of peaks, when the aggregator refreshes rarely, sources become saturated very often and the $2Steps$ strategy focuses itself on refreshing those saturated ones. Predicting when a source publishes $W_s = 20$ items in the case of sources with regular peaks works well both with the periodic and the single



(a) Peaks

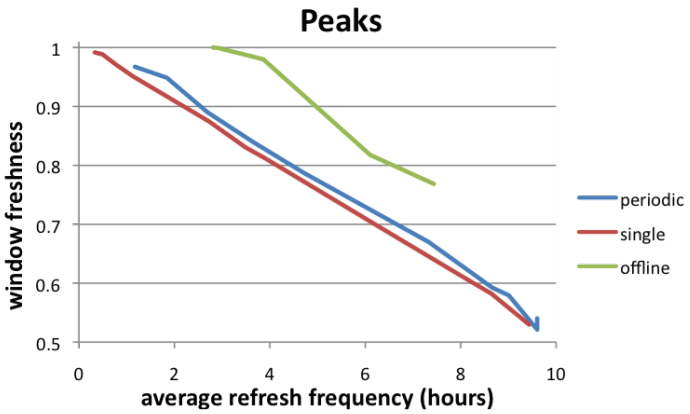


(b) Uniform

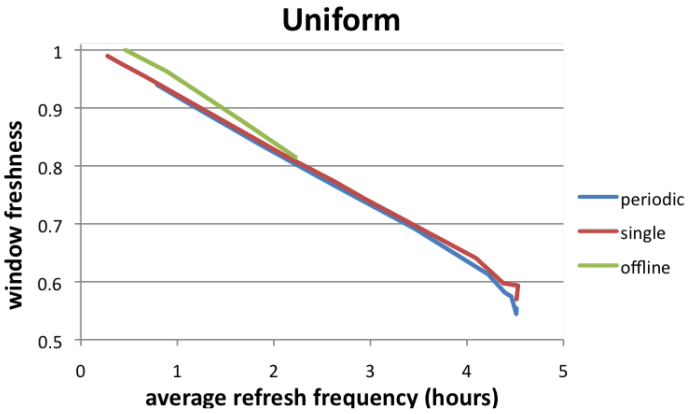


(c) Waves

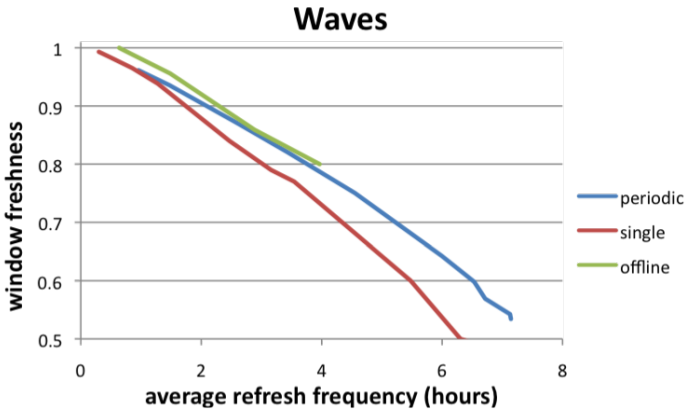
Figure 7.9: Feed completeness



(a) Peaks



(b) Uniform



(c) Waves

Figure 7.10: Window freshness

variable publication model, because in this case the precision offered by the periodic model (that knows exactly at which point in time each item was published) is useless. All these make that both periodic and single variable publication estimation give similar results in terms of feed completeness and window freshness for the peaks in case of rare refreshes. When sources are refreshed more often and there are less saturated sources, periodic publication estimation give better results. In the case of wavy publication behavior, periodic estimation outperforms the single variable one because of the information accuracy it provides, no matter how often the sources are refreshed. For the uniform sources, the same conclusions as for the uniform random strategy hold for the $2Steps$ refresh strategy.

7.3.4 Discussion

Experimental results illustrate the high influence of the used publication model together with its corresponding estimation technique on the overall quality of the refresh process. And vice versa, we have shown that the refresh frequency used by the strategy has an important impact on the quality of the estimation process. Furthermore, saturation has a highly negative impact: if refreshes are not done often enough and items are lost, the estimation process uses inaccurate data for updating the model. In this case, a possible solution is the separation between the estimation from the refresh process of the crawling module, by separating the bandwidth resources needed for the two processes.

When the refresh strategy has big limitations in terms of bandwidth usage, online estimation does not represent a reliable solution. One viable solution is to allocate more bandwidth resources for learning a publication profile during a limited period of time and then to use the learnt model to refresh the sources, without updating it. This gives good results for feeds (or queries on feeds) that do not change their publication behavior in time, but it is not a reliable technique when dealing with highly dynamic web resources. Moreover, several such learning periods may be repeated periodically to update the source publication profiles. Since a refresh strategy is based on a publication model and the estimation of the publication model depends on the refresh frequency used by the refresh strategy, the usage of online change estimation techniques represents a challenge.

7.4 Statistical Estimation Model

In this section we present a formal method for estimating a source publication frequency based on the maximum likelihood estimation method. We make the hypothesis that the publication activity of a source represents a homogeneous Poisson process, characterized by the rate parameter λ. Informally, the maximum likelihood estimator computes the value of λ which has the highest probability of producing an observed set of events.

7.4.1 Homogeneous Poisson Process

If the expected number of published items in the time unit interval is λ, then the probability that there are exactly k new items in the time interval Δt is equal to:

$$P_\lambda(k, \Delta t) = \frac{(\lambda \Delta t)^k \cdot e^{-\lambda \Delta t}}{k!}$$

A source has a publication window of size W_s and each update appends a newly published item and evicts the oldest item from the publication window that was appended W_s updates earlier, such that at any given time there are only W_s items available in the source publication window. In this case, the probability that the lifetime of an item in the publication window is inferior to the time interval Δt is equal to:

$$P(lifetime \leq \Delta t) = 1 - \sum_{i=0}^{W_s - 1} P_\lambda(i, \Delta t) = 1 - \sum_{i=0}^{W_s - 1} \frac{(\lambda \Delta t)^i \cdot e^{-\lambda \Delta t}}{i!}$$

If source s was last refreshed at time moment T_r then the window divergence function at time moment t is:

$$Div_A(s, t, T_r) = \begin{cases} W_s & \textbf{with the probability } 1 - \sum_{i=0}^{W_s - 1} \frac{(\lambda(t - T_r))^i \cdot e^{-\lambda(t - T_r)}}{i!} \\ k & \textbf{with the probability } \frac{(\lambda(t - T_r))^k \cdot e^{-\lambda(t - T_r)}}{k!}, \forall k < W_s \end{cases} \quad (7.5)$$

7.4.2 Publication Profile

Each time the aggregator refreshes a source s at time moment T_r, it observes $x^r = Div(s, T_r, T_{r-1})$, the number of new items published since the last refresh at T_{r-1}. The aggregator collects these observations into a *publication profile*, that consists of a sequence of (time interval ΔT_r, divergence x^r) pairs, where $\Delta T_r = T_r - T_{r-1}$ represents the time interval between two consecutive refreshes. The size of the publication profile of a source kept by an aggregator is limited to the most recent n observations.

An example of a publication profile is: $pubProfile =< (10, 5), (15, 1), (8, 3), (20, 8) >$. Supposing that time is measured in minutes, the publication profile of the source consists of 4 consecutive observations and it indicates that it published 5 items during 10 minutes, 1 item in 15 minutes, 3 items in 8 minutes and lastly, 8 items in 20 minutes time.

7.4.3 Maximum Likelihood Estimation

We use the method of the *maximum likelihood* that for a fixed set of data (i.e. a source publication profile) and an underlying statistical model (i.e. homogeneous Poisson process), it selects the value of the model parameter (i.e. λ) that produces a distribution that gives the observed data the greatest probability.

We denote by $\mathcal{L}(\lambda|pubProfile)$ the likelihood of the Poisson rate parameter value λ given the observed source publication profile $pubProfile$ which is equal to the probability of the observed source publication profile $pubProfile$ given the parameter value λ:

$$\mathcal{L}(\lambda|pubProfile) = \prod_{i=1}^{n} P_\lambda(x_i^r, \Delta T_{r_i})$$

where $P_\lambda(x_i^r, \Delta T_{r_i})$ represents the probability of observing x_i^r new items published during the time interval ΔT_{r_i} and $(\Delta T_{r_i}, x_i^r) \in pubProfile$.

We further compute the log-likelihood function as, using Equation 7.5:

$$\begin{aligned}
\ln \mathcal{L}(\lambda|pubProfile) &= \sum_{i=1}^{n} \ln P_\lambda(x_i^r, \Delta T_{r_i}) \\
&= \sum_{i=1}^{n} \ln \frac{(\lambda \Delta T_{r_i})^{x_i^r} \cdot e^{-\lambda \Delta T_{r_i}}}{x_i^r!} \Big/_{\text{s.t. } x_i^r < W_s} \\
&+ \sum_{i=1}^{n} \ln \left(1 - \sum_{j=0}^{W_s-1} \frac{(\lambda \Delta T_{r_i})^j \cdot e^{-\lambda \Delta T_{r_i}}}{j!}\right) \Big/_{\text{s.t. } x_i^r = W_s}
\end{aligned}$$

The goal of our estimation method is to find the value λ of the source publication frequency that maximizes the log-likelihood function $\ln \mathcal{L}(\lambda|pubProfile)$. We do that using the gradient descent method. We compute the gradient of the log-likelihood function and we search the λ value for which the gradient function is zero:

$$\frac{\partial\big(ln\mathcal{L}(\lambda|pubProfile)\big)}{\partial\lambda} = 0$$

$$\begin{aligned}
\frac{\partial\big(ln\mathcal{L}(\lambda|pubProfile)\big)}{\partial\lambda} &= \sum_{i=1}^{n} \frac{x_i^r \cdot \Delta T_{r_i}}{\lambda} \Big/_{\text{s.t. } x_i^r < W_s} \\
&+ \sum_{i=1}^{n} \frac{\sum_{j=0}^{W_s-1} \frac{(\lambda \Delta T_{r_i})^j \cdot e^{-\lambda \Delta T_{r_i}}}{j!} \cdot \frac{\lambda \Delta T_{r_i} - j}{\lambda}}{1 - \sum_{j=0}^{W_s-1} \frac{(\lambda \Delta T_{r_i})^j \cdot e^{-\lambda \Delta T_{r_i}}}{j!}} \Big/_{\text{s.t. } x_i^r = W_s}
\end{aligned}$$

The actual value of the gradient $\frac{\partial\big(ln\mathcal{L}(\lambda|pubProfile)\big)}{\partial\lambda}$ can be computed given a certain λ^k value. The iterative algorithm of the gradient descent method is presented in the Algorithm 7.1.

The algorithm starts from an initial value of $\lambda = \lambda^0$ and updates it during several iterations. Since we want to maximize the log-likelihood function, at iteration k we decrease the value of λ^k if the gradient value is negative or increase it if the gradient value is positive with a term proportional to the gradient of the log-likelihood at the current point

Algorithm 7.1 Lambda Estimation with Gradient Method

Input: $\lambda^0, \epsilon, minLambda, stopThreshold, maxIterations$
$\lambda^k = \lambda^0$, $k = 0$
while $\frac{|\lambda^{k+1} - \lambda^k|}{\lambda^k} > stopThreshold$ AND $k < maxIterations$ **do**
  $\lambda^{k+1} = \lambda^k + \epsilon \cdot \frac{\partial\left(ln\mathcal{L}(\lambda|pubProfile)\right)}{\partial\lambda}\Big/_{\lambda^k}$
  if $\lambda^{k+1} \leq 0$ **then**
    $\lambda^{k+1} = minLambda$
  end if
  $k = k + 1$
end while

(computed for $\lambda = \lambda^k$). We also prevent the value of λ from becoming negative by limiting it to a minimum positive value $minLambda$. We iterate until at least one of the two exit conditions are met: either the value of λ has converged ($|\lambda^{k+1} - \lambda^k|/\lambda^k \leq stopThreshold$) or we have exceeded the maximum number of allowed iterations.

7.5 Conclusion

In this chapter we have investigated problems related to an RSS aggregator that retrieves information from multiple RSS feed sources automatically. In particular, we have proposed and studied two online estimation methods that correspond to two different models of the source publication activity. We tested the online estimation methods in cohesion with di?erent refresh strategies. We compared these methods for different publication activity shapes and we highlighted the challenges imposed by the application context.

Conclusion and Future Work

As the Internet grows larger and the Web 2.0 services get more popular, the online web content becomes more diverse and dynamic. However, such an increase in the quantity of the new information generated every day might easily get overwhelming for the users. This problem has been partially solved by the introduction of the RSS [rssa] and Atom [ato] data formats, that have become the de-facto standards for efficient information dissemination. In this context, the role of feed content aggregators that help users manage their subscriptions becomes even more important.

This dissertation studied some problems encountered when designing a content-based feed aggregation system and focused on different points: the conception of optimal refresh strategies that maximize the aggregation quality with an efficient use of bandwidth resources and online change estimation techniques developed for dynamic RSS source publication models. More specifically, in this dissertation we addressed the following challenges and proposed the following contributions.

Contributions

Content-based feed aggregation system. In chapter 1 we started by describing RoSeS, a feed aggregation system [HAA10, HAA11] that gathers web content coming from different online RSS feed sources, applies content-based aggregation queries and delivers the resulting personalized aggregated feeds to users or stores them in a database for later reuse. We exemplified content-based aggregation queries over a set of RSS sources as stateless continuous queries computing a filtered union of source feeds. We proposed a feed aggregation model and discussed feeds from different points of view. We analyzed two different semantics of a feed which can be seen either as a continuous stream or as a limited size publication window of items. Last, we described an aggregation network model and proposed a topology generation method inspired by the Internet structure.

Feed aggregation quality measures. For evaluating the quality of aggregated feeds, we proposed in chapter 2 different types of aggregation quality measures: feed completeness and window freshness. Both quality measures reflect item loss, but in different ways.

Feed completeness is based on the stream semantics, characterizes long-term aggregation processes and considers item loss as being definitive, as these items are lost forever. Window freshness is based on the window semantics and considers the aggregation process from a short-term point of view, being concerned with temporary item loss, as these items still can be retrieved at refresh time.

Best effort refresh strategies for RSS feeds. The refresh strategy used by a content-based feed aggregation system impacts directly the quality of the newly generated aggregated feeds. In chapter 4 we proposed a two steps best effort refresh strategy for RSS feeds [HAA10, HAA11] based on the Lagrange multipliers method. Best effort strategies achieve maximum aggregation quality (in terms of feed completeness and window freshness) compared with all other policies that perform an equal average number of refreshes. While the bandwidth resources are sufficient to keep a high refresh frequency such that feed sources do not reach their saturation point before being refreshed, our refresh strategy maximizes both feed completeness and window freshness. As bandwidth resources become more limited and feed sources risk saturation, our refresh strategy focuses on minimizing permanent items loss and thus, maximizing feed completeness. Instead of solving the equations introduced by the Lagrange multipliers method, we reformulated the solution based on a utility function and a refresh threshold parameter, that controls the overall refresh rate and that can be dynamically adjusted if the feed source publication behaviors vary in time. One major advantage of our refresh strategy is that it does not need to pre compute a fixed refresh schedule, but it is able to assign the next time moments for refreshing the feed sources immediately prior to these time moments. This robustness property makes it compatible with online change estimation methods for modeling the feed activity, that update the source publication models "on the fly".

RSS feeds evolution characteristics. Understanding how web resources evolve in time is important for conceiving tools designed to ease the interaction between people and the dynamic web content. In chapter 6 we presented an analysis of general characteristics with a focus on the temporal dimension of real RSS feed sources, using data collected over four weeks from more than 2500 RSS feeds. More exactly, we examined the publication activity, publication periodicity and publication shapes [HAA12], looking into how much new information is published daily by the RSS feeds, when exactly it is published and if there are any publication patterns. First, the publication activity is concerned with the intensity of the source publication process, classifying feeds from very slow to very productive. Second, the publication periodicity searches for daily repetitive publication behaviors of the feed sources. And third, the publication shapes analyzes the daily publication activity forms of the periodic feeds and classies them in three categories: peaks, uniform and waves.

Online change estimation techniques. Inspired by the observations made on the real RSS feeds publication behavior, in chapter 7 we proposed two different source publication

models that reflect different types of feed publication activities. In order to keep these publication models up to date as source publication behaviors change, we also studied different online change estimation techniques [HAA12, HAA11] with the particular goal to improve the refresh strategies used by the RSS aggregators. We introduced the single variable and the periodic source publication models and analyzed methods for estimating them online in order to reflect the changes in the real feed publication activities. We also detailed a hybrid publication model and discussed a dynamic algorithm for adjusting the value of the estimation smoothing parameter. Finally, we integrated the proposed online change estimation techniques with our best effort feed refresh strategy and tested them on real source feeds having different types of publication shapes (peaks, uniform and waves).

We now briefly discuss some potential areas for future work and directions in which our work may envisage expanding.

Perspectives and Future Work

There are different ways in which our proposed model can be improved, such as the current solutions might be optimized and generalized. We also introduce a set of challenging possibilities to validate and extend our work in different application contexts, such as personalization applications, distributed networks and social medias.

Model extension. In this dissertation we consider the case of stateless continuous aggregation queries computing a filtered union of source feeds. We can also envisage expanding the class of aggregation queries we consider by adding joins among source feeds, as described in [TATV11].

One possible way to improve the change prediction is to take into account the correlations between different feed publication activities, if any. For example, as a scoop gets published in a news feed, it is expected that other news feeds publish on the same topic. In this case, we may use sampling techniques: when we want to refresh the feeds, we first check for changes only a couple of "sample" feeds and refresh the remainder only if the sample feeds have changed.

We can extend our refresh strategy to take into account nonuniform refresh costs. This is especially interesting when working in environments where the cost to refresh objects is not uniform, possibly because they have different sizes. A possible way to account for nonuniform refresh costs is to consider a source importance function that includes a factor inversely proportional to the cost.

Our aggregation model and refresh strategy can also be extended by taking weights or importance functions into consideration. For example, the divergence function $Div(s, q, t, T_r)$ that represent the total number of items relevant to query q published by the source s in the time interval $(T_r, t]$ may also depend on the importance of the source s and that of the aggregation query q. On the one hand, we consider the importance of a source

s. Its value might be fixed according to various criteria, including but not limited to data quality, content provider authority (e.g. PageRank [BP98]), financial considerations, probability of access and popularity reasons. On the other hand, we consider the weight of an aggregation query q. There are multiple ways of establishing its value, many of them based on the importance of query terms, using statistical language modeling techniques. These weight functions can be composed to obtain an overall importance function for an aggregation query q applied to feed source s.

Personalization applications. Furthermore, the importance functions considered may be used in the context of the personalization applications. Besides the already mentioned ways of fixing the weight values, one can imagine users express trust scores for different sources and interest levels for different topics or aggregation queries that can be included in the importance values computation.

In the same personalization context, an interesting alternative with a very practical side is to formulate the optimization problem differently. Instead of maximizing aggregation quality, given certain bandwidth constraints, we can allow users to express desired minimum aggregation quality levels that have to be assured and for which they are willing to pay and have as goal the bandwidth consumption minimization.

When dealing with a large number of source feeds and even more when taking user personalization into account, the optimization problem dimension grows significantly. One possible solution to reduce costs is to benefit from the advantages of clustering [XI05]. For example, feeds that have similar source publication profiles may be grouped in the same cluster. All sources in the same cluster are characterized by an average source publication profile, that results in important memory space savings. Furthermore, as the publication profile of a source changes in time, it is likely that the source should be moved from one cluster to another. Therefore, the clusters should be dynamically updated to reflect the current source publication profiles.

Distributed web syndication network. Another possible direction we consider as future work, one that opens many new possibilities, is to adapt our feed aggregator and its refresh strategy we proposed for the client-server case to a more general context: content based feed aggregation in distributed web syndication systems. A first step in this direction was made in Section 1.6 by introducing a feed aggregation network model and a topology generation method, inspired by the Internet structure, that connects aggregator nodes together in a distribution graph, based on their subscription queries. Although our refresh strategy was conceived to maximize aggregation quality in a client-server scenario, it would be interesting to study its impact in the distributed setting, when refreshes would occur between feed aggregators on consecutive levels, with a possible cascading effect. Furthermore, the model can be enhanced by exploiting some new possibilities offered by the distributed setting, such as introducing nodes aware of some graph based knowledge or collaborative nodes that exchange useful information in order to achieve a global

optimization goal.

Social medias. Going one step farther, we consider the context of today's social medias that propose advanced personalization services through user search queries and the possibility to define complex score functions on a very dynamic and distributed content. The world of social media is a very complex one, with many different available types, such as collaborative projects (e.g., Wikipedia [wika]), blogs and microblogs (e.g., Twitter [twi]), content communities (e.g., YouTube [you]) or social networking sites (e.g., Facebook [fac]) that can be furthermore integrated via social network aggregation platforms. Moreover, the users can act both as producers and seekers of information and the social network, the data and the personalized search queries can change at any moment. The challenge in such a rich application context is to extend our model, refresh strategies and change estimation techniques in order to optimize the synchronization of the online content and the interaction between individuals and communities.

Part IV

Appendix

Appendix A

Résumé de la Thèse en Français

Depuis son apparition il y a vingt ans, le Web a beaucoup évolué, d'un système de publication hypertexte vers une plateforme de participation et collaboration des utilisateurs [O'R05], aussi appelée *Web 2.0*. Les services Web 2.0 facilitent la création et la gestion du contenu disponible en ligne. Beaucoup d'applications Web 2.0 sont apparues pour permettre aux utilisateurs de gérer, collaborer et partager sur le Web leurs informations personnelles. Comme exemple nous pouvons citer les sites de réseaux sociaux pour rester en contact avec les amis, la famille et les partenaires professionnels, les galeries en ligne pour les photos et vidéos ou les blogs pour partager les agendas et les journaux de voyage, comme le service de publication des blogs Blogger [bloa], l'encyclopédie en ligne Wikipedia [wika], le partage de photos Flickr [fli], le partage de marque-pages web Delicious [del], le site de partage social de liens web Digg [dig], le réseau social Facebook [fac] et le service de microblogging Twitter [twi].

Cette grande accessibilité spécifique aux applications Web 2.0 a généré une grande explosion de documents accessibles en ligne. En 2006, le magazine Time a présenté UGC (User Generated Content le contenu généré par les utilisateurs) comme la "Personne de l'Année" [tim], faisant référence à tous les créateurs individuels de contenu en ligne qui contribuent aux médias générés par les utilisateurs. D'autres exemples qui illustrent la *dynamique en ligne* générale et la *croissance rapide* du contenu web incluent l'encyclopédie collaborative en ligne Wikipedia [wika] qui signale qu'elle enregistre presque 8000 mises à jour par heure pour les articles en anglais et 2000 pour ceux en français [wikb]. Après analyse de la croissance de la blogosphère, NM Incite [nmi] a découvert plus de 181 millions de blogs dans le monde fin 2011, quand cinq ans plus tôt, en 2006, il n'y en avait que 36 millions. L'agrégateur de news libre Google News [goob] offre des informations agrégées à partir de plus de 25000 éditeurs [gooc] du monde entier et de presque 4500 sites en anglais, chacun avec sa propre activité indépendamment de la publication. De plus, il y a de nombreux services qui offrent une surveillance et mise à jour en temps réel et une analyse des valeurs de la bourse, domaine qui est connu pour sa nature fortement dynamique.

En raison de la croissance rapide des sources d'informations en ligne, il est difficile pour

un utilisateur de rester informé de toutes les nouveautés du Web. Ce problème a été partiellement résolu avec l'introduction des *formats de données RSS* [rssa] et *Atom*[ato], devenus le standard de diffusion d'informations continues. Pour aider les utilisateurs à accéder au contenu nouveau diffusé à travers les flux RSS auxquels ils sont abonnés et rester à jour avec celui-ci, de nombreux *agrégateurs RSS* sont récemment apparus et gagnent en popularité, comme Bloglines [blob], Google Reader [good], RSScache [rssc] ou Technorati [tec].

L'intégration de flux RSS et Atom dans les interfaces et portails web permet aux utilisateurs de rester informés et de suivre "en direct" l'évolution de nombreux sites d'intérêt. Pratiquement, un flux RSS est un document XML qui contient une liste d'items et la publication de chaque item correspond à un changement sur le site associé au flux RSS. Les items dans la liste sont généralement limités aux derniers articles publiés. En s'abonnant à des flux RSS, les utilisateurs peuvent suivre les dernières modifications sans visiter le site et protègent aussi leur données personnelles en évitant de s'inscrire à la newsletter du site. La Figure A.1 donne des exemples de sites typiques qui utilisent le RSS.

Figure A.1: Utilisation du RSS sur des différents types de sites web

En règle générale, il existe deux options opposées pour la diffusion d'une information d'un serveur vers un client à travers un réseau. La première option consiste à transmettre les informations à la demande explicite du client (mode "pull"). La deuxième option se situe à l'opposé : le serveur transmet les nouvelles informations au client qui a effectué une seule demande initiale de souscription (mode "push"). Dans le contexte du Web, le protocole pull a été largement adopté pour plusieurs raisons. Tout d'abord, l'adoption du mode push complique la communication, en nécessitant un ensemble de règles et de conventions qui doivent être respectées par les fournisseurs de contenu, mais aussi par les agrégateurs. Ainsi, en utilisant le protocole push, les fournisseurs de contenu sont forcés de maintenir

une liste des abonnées qui peut devenir très grande avec le temps et en conséquence, très difficile à gérer. D'un autre côté, l'utilisation des protocoles de type pull, tels que le HTTP ou le XML-RPC, préserve la grande autonomie des fournisseurs de contenu, mais aussi des agrégateurs. Et finalement, adopter le protocole push demande l'existence et l'entretien d'une relation de confiance entre les fournisseurs de contenu et les agrégateurs de contenu, qui est très difficile de gérer et très peu probable dans le Web réel. Le protocole pull fonctionne sans que les fournisseurs de contenu soient conscients des agrégateurs abonnés. Une discussion plus détaillée et des références d'état de l'art sur les protocoles pull et push sont présentés dans la Section 3.1.

Quand nous examinons l'échange de messages du point de vue d'un serveur web il n'y a pas de différence entre les flux RSS, les pages web ou toute autre ressource web. Toutes ces ressources sont accessibles en utilisant le protocole HTTP en mode pull. Ceci signifie en particulier que c'est le client qui doit décider quand il veut rafraîchir une ressource pour prendre en compte des changements éventuels. Donc les agrégateurs, ainsi que les moteurs de recherche, affrontent le même type de défi : décider du moment optimal pour rafraîchir chaque ressource.

Notre travail de recherche est situé dans le contexte du projet RoSeS (Really Open Simple and Efficient Syndication) [rosa]. Ce projet a pour objectif la définition d'un ensemble de services de syndication de ressources web et des outils pour la localisation, l'interrogation, la génération, la composition et la personnalisation de flux RSS disponibles sur le Web. Un agrégateur RoSeS représente un système de partage de flux RSS dans lequel les utilisateurs peuvent enregistrer des requêtes sur le contenu de flux RSS dont les résultats sont des nouveaux flux RSS.

Placé dans ce contexte, l'*objectif* de cette thèse est d'améliorer la qualité d'agrégation en concevant des *stratégies de rafraîchissement* et des *méthodes d'estimation en ligne du changement de contenu* adaptées aux sources de flux RSS hautement dynamiques. Malgré la simplicité apparente, les systèmes d'agrégation de flux ont de nombreux défis intrinsèques :

- **Large échelle.** Non seulement le Web est très grand, mais il évolue en permanence, il en va de même pour les flux RSS disponibles sur les sites web. Par conséquent, les agrégateurs de flux doivent supporter un débit très fort et faire face à un très grand nombre de fournisseurs et consommateurs de contenu de type flux en obtenant une haute qualité d'agrégation.
- **Contenu hautement dynamique.** Comme toutes les ressources web, les flux RSS évoluent indépendamment de leurs clients et peuvent changer soudainement d'activité de publication. Même si un agrégateur supporte la syndication à large échelle, il ne pourra jamais suivre tous les changements dynamiques de tous les flux.
- **Décision de rafraîchissement.** Afin d'assurer le bon fonctionnement d'un agrégateur, il doit employer une *stratégie de rafraîchissement* intelligente qui décide quand rafraîchir chaque source de flux afin de garantir la *maximisation de la qualité d'agrégation*, en utilisant un *coût minimal*.

- **Ressources limitées.** Non seulement les crawleurs doivent respecter les politiques de politesse pour ne pas imposer trop de charge sur les fournisseurs de contenu, mais ils sont aussi contraints par leur propres limitations internes de ressources, tels que la bande passante, l'espace de stockage, la mémoire ou les ressources de calcul, tout en minimisant le coût total.
- **La perte d'informations.** Un flux RSS est un document XML avec les derniers articles publiés. Le nombre d'articles est limité et une fréquence de rafraîchissement trop faible peut mener à la *perte d'informations*. Ceci ne réduit pas seulement la qualité des résultats d'agrégation (*complétude*), mais rend également l'estimation des fréquences de publications plus difficile.
- **Obsolescence de l'information.** Le contenu de flux RSS est souvent lié aux événements du monde réel et la valeur informative des articles publiés se dégrade avec le temps. Une *stratégie de rafraîchissement* optimale employée sur un agrégateur de flux devrait être capable de retrouver rapidement le contenu publié récemment, pour garder un niveau élevé de *fraîcheur* d'agrégation.
- **Connaissances incomplètes.** La nature dynamique du contenu web mène à la nécessité de mettre à jour en continu l'estimation de la fréquence de publication, en utilisant des *techniques d'estimation en ligne*. Vu que les modèles de publication des sources sont mises à jour seulement au moment du rafraîchissement, les estimateurs en ligne doivent pouvoir travailler avec des *connaissances incomplètes* sur l'historique de changements des données, parce qu'ils peuvent ne pas savoir exactement combien, quand et à quelle fréquence une source produit de nouveaux articles.
- **Intervalles d'estimation irréguliers.** Il y a pleins d'applications web dans lesquelles les sources de données ne sont pas rafraîchis à intervalles de temps réguliers. Le moment précis d'accès est décidé par la stratégie de rafraîchissement, conçue pour optimiser certaines mesures de qualité en utilisant un coût minimal. Les *intervalles irréguliers de rafraîchissement* et *l'historique incomplet des changements* rendent très difficile le processus d'estimation.

Contributions

Dans cette thèse, nous nous proposons de résoudre les défis posés aux agrégateurs RSS décrits ci-dessus. Les principaux problèmes présentés et étudiés dans cette thèse, ainsi que nos différentes contributions essaient de répondre aux questions suivantes :

Comment concevoir un système d'agrégation de flux basé sur le contenu et définir ses caractéristiques principales ? Les utilisateurs d'un agrégateur RSS désirent rester informés des dernières nouveautés sur les sujets et les sources des flux RSS auxquels ils s'intéressent. Pour cela, ils enregistrent des requêtes d'agrégation différentes sur un ensemble de flux RSS reflétant leurs intérêts. Les résultats des requêtes d'agrégation sont ensuite rendus disponibles aux utilisateurs comme des flux RSS fraîchement créés. Nous

proposons une architecture et un modèle déclaratif pour un système d'agrégation de flux basé sur le contenu avec une sémantique précise pour les flux et les requêtes d'agrégation.

Comment évaluer la qualité d'un système d'agrégation de flux et quelles sont les mesures de qualité adaptées ? Pour évaluer la qualité des résultats d'un système d'agrégation, nous proposons deux mesures différentes : la "complétude flux" et la "fraîcheur fenêtre". La complétude flux évalue le taux de couverture des articles récupérés qui correspondent à une requête d'agrégation. La fraîcheur fenêtre, mesurée à un certain moment, reflète la fraction des articles de flux récemment publiés qui n'ont pas encore été retrouvé. Une fraîcheur fenêtre maximale montre que le flux RSS résultat d'une requête d'agrégation est à jour avec toutes ses sources de flux RSS. Les mesures qualitatives proposées (la complétude flux et la fraîcheur fenêtre) ont été conçues pour décrire la qualité des flux produits dans le cadre d'un système d'agrégation flux basé sur le contenu.

Quand est-ce que le crawleur d'un système d'agrégation de flux doit rafraîchir les sources de flux RSS et qu'est-ce qui rend une stratégie de rafraîchissement optimale ? Nous proposons une stratégie de rafraîchissement "best effort" basée sur la méthode d'optimisation des multiplicateurs de Lagrange [Ste91], qui maintient un niveau optimal de qualité d'agrégation, pour un coût moyen fixé. Cette stratégie maximise la complétude flux et la fraîcheur fenêtre, récupère les nouveaux articles à temps, évitant la perte d'informations. La stratégie proposée est accompagnée d'une évaluation expérimentale extensive, qui teste sa performance et robustesse en comparaison avec d'autres stratégies de rafraîchissement.

Comment caractériser l'activité de publication d'un flux RSS ? Pour mieux comprendre le comportement de publication des flux RSS réels, nous proposons une analyse des caractéristiques générales qui se concentre sur la dimension temporelle des sources réelles de flux RSS, en utilisant des données collectées pendant quatre semaines et provenant de plus de 2500 flux RSS. Pour commencer, nous analysons l'intensité de l'activité de publication de flux et les caractéristiques de périodicité quotidienne. De plus, nous classifions les sources de flux en fonction de trois formes différentes de publication : par pics, uniforme et par vagues.

Comment estimer le nombre d'items publiés par un flux pendant une période de temps ? Inspirés par les observations faites sur l'activité réelle de publication de flux RSS, nous proposons deux modèles qui reflètent différents types d'activités de publication de flux. De plus, nous étudions deux méthodes d'estimation en ligne qui correspondent aux modèles de publication mis à jour en continu pour suivre les changements d'activités réelles de publication de flux. Nous présentons une évaluation expérimentale des méthodes d'estimation en ligne intégrées à différentes stratégies de rafraîchissement et nous analysons leur efficacité sur des sources avec différentes activités de publication.

Organisation du Mémoire

Ce mémoire est organisé comme suit. Le Chapitre 1 introduit quelques notions fondamentales du domaine de l'agrégation de flux basée sur le contenu. Nous décrivons l'architecture de notre système d'agrégation pour mieux comprendre les méthodes proposées et nous introduisons un modèle formel d'agrégation de flux basée sur le contenu. Le Chapitre 2 définit deux mesures de qualité orientées agrégation : la complétude flux et la fraîcheur fenêtre. Nous expliquons le problème de saturation spécifique aux flux RSS et nous présentons une mesure de perte d'information. Le Chapitre 3 détaille le problème général de crawling web, en introduisant quelques facteurs clés et objectifs ciblés par les crawleurs web qui influencent leurs stratégies de crawling. Nous nous focalisons ensuite sur le problème spécifique du rafraîchissement de flux RSS et nous présentons différentes approches proposées dans la littérature sur les stratégies de rafraîchissement de flux. Le Chapitre 4 propose une stratégie de rafraîchissement "best effort" qui maximise la qualité d'agrégation (la complétude flux et la fraîcheur fenêtre) avec un coût moyen fixé d'avance. A la fin du chapitre, nous présentons les expériences qui testent notre stratégie en comparaison avec d'autres stratégies de rafraîchissement.

Dans le Chapitre 5 nous décrivons un modèle d'activité de publication de flux RSS. Ensuite nous examinons des modèles de changements du Web proposés pour les pages web et pour les flux RSS. Nous comparons aussi différentes méthodes d'estimation de changements du Web et nous nous concentrons sur les méthodes d'estimation en ligne. Le Chapitre 6 introduit une analyse approfondie de flux RSS réels, avec un accent sur la dimension temporelle de leur activité de publication. Nous nous sommes intéressés à l'activité, la périodicité et les formes de publication. Le Chapitre 7 propose deux modèles pour estimer le nombre d'items publiés par un flux pendant une période de temps, ainsi que des méthodes pour mettre à jour ces modèles d'estimation suivant l'activité réelle de flux. Nous présentons une évaluation expérimentale des méthodes d'estimation en ligne.

Enfin le chapitre de conclusion résume nos différentes contributions et soulève un ensemble de problèmes considérés comme importants pour les perspectives de recherche.

List of Figures

List of Algorithms

Bibliography

[ABP10] George Adam, Christos Bouras, and Vassilis Poulopoulos. Efficient extraction of news articles based on RSS crawling. In *International Conference on Machine and Web Intelligence*, 2010.

[AFZ97] Swarup Acharya, Michael J. Franklin, and Stanley B. Zdonik. Balancing Push and Pull for Data Broadcast. In Joan Peckham, editor, *SIGMOD Conference*, pages 183–194. ACM Press, 1997.

[APC03] Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In *WWW*, pages 280–290, 2003.

[ato] The Atom Publishing Protocol. http://tools.ietf.org/html/rfc5023.

[BBB+03] Noam Berger, Béla Bollobás, Christian Borgs, Jennifer T. Chayes, and Oliver Riordan. Degree Distribution of the FKP Network Model. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 725–738. Springer, 2003.

[BC00] Brian E. Brewington and George Cybenko. How dynamic is the Web? *Computer Networks*, 33(1-6):257–276, 2000.

[BDH+95] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest Information Discovery and Access System. *Computer Networks and ISDN Systems*, 28(1&2):119–125, 1995.

[BGR06] Laura Bright, Avigdor Gal, and Louiqa Raschid. Adaptive pull-based policies for wide area data delivery. *ACM Trans. Database Syst.*, 31(2):631–671, 2006.

[bloa] Blogger. http://www.blogger.com/.

[blob] Bloglines. http://www.bloglines.com/.

[BMvD07] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A Comparison of Push and Pull Techniques for AJAX. In Shihong Huang and Massimiliano Di Penta, editors, *WSE*, pages 15–22. IEEE Computer Society, 2007.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[BYCMR05] Ricardo A. Baeza-Yates, Carlos Castillo, Mauricio Marín, and Andrea Rodríguez. Crawling a country: better strategies than breadth-first for web page ordering. In Allan Ellis and Tatsuya Hagino, editors, *WWW (Special interest tracks and posters)*, pages 864–872. ACM, 2005.

[CGM00a] Junghoo Cho and Hector Garcia-Molina. Synchronizing a Database to Improve Freshness. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD Conference*, pages 117–128. ACM, 2000.

[CGM00b] Junghoo Cho and Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB*, pages 200–209. Morgan Kaufmann, 2000.

[CGM02] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 124–135, New York, NY, USA, 2002. ACM.

[CGM03a] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for Web crawlers. *ACM Trans. Database Syst.*, 28(4):390–426, December 2003.

[CGM03b] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Trans. Internet Technol.*, 3(3):256–290, August 2003.

[CGMP98] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient Crawling Through URL Ordering. *Computer Networks*, 30(1-7):161–172, 1998.

[Cha04] C. Chatfield. *The Analysis of Time Series: An Introduction.* CRC Press, 2004.

[CLW98] E. G. Coffman, Zhen Liu, and Richard R. Weber. Optimal robot scheduling for Web search engines. *Journal of Scheduling*, 1(1):15–29, 1998.

[com] Complete RSS. http://completerss.com/.

[CvdBD99] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific Web resource discovery. In *Proceedings of the eighth international conference on World Wide Web*, WWW '99, pages 1623–1640, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[CY08] Badrish Chandramouli and Jun Yang. End-to-end support for joins in large-scale publish/subscribe systems. *PVLDB*, 1(1):434–450, 2008.

[del] Delicious. http://www.delicious.com/.

[dig] Digg. http://digg.com/.

[EMT01] Jenny Edwards, Kevin S. McCurley, and John A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *WWW*, pages 106–113, 2001.

[fac] Facebook. http://www.facebook.com/.

[FCV09] Dennis Fetterly, Nick Craswell, and Vishwa Vinay. The impact of crawl policy on web search effectiveness. In James Allan, Javed A. Aslam, Mark Sanderson, ChengXiang Zhai, and Justin Zobel, editors, *SIGIR*, pages 580–587. ACM, 2009.

[fee] Feedmil. http://www.feedmil.com/.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-law Relationships of the Internet Topology. In *SIGCOMM*, pages 251–262, 1999.

[FKP02] Alex Fabrikant, Elias Koutsoupias, and Christos H. Papadimitriou. Heuristically Optimized Trade-Offs: A New Paradigm for Power Laws in the Internet. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 110–122. Springer, 2002.

[fli] Flickr. http://www.flickr.com/.

[FZ98] Michael J. Franklin and Stanley B. Zdonik. "Data In Your Face": Push Technology in Perspective. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD Conference*, pages 516–519. ACM Press, 1998.

[GGLNT04] Daniel Gruhl, Ramanathan V. Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW*, pages 491–501. ACM, 2004.

[gig] Giga Alert. http://www.gigaalert.com/.

[gooa] Google Alerts. http://www.google.com/alerts.

[goob] Google News. https://news.google.com/.

[gooc] Google News - more than 25,000 publishers. http://googlenewsblog.blogspot.fr/2009/12/same-protocol-more-options-for-news.html.

[good] Google Reader. http://www.google.com/reader.

[GS96] James Gwertzman and Margo I. Seltzer. World Wide Web Cache Consistency. In *USENIX Annual Technical Conference*, pages 141–152, 1996.

[HAA10] Roxana Horincar, Bernd Amann, and Thierry Artières. Best-Effort Refresh Strategies for Content-Based RSS Feed Aggregation. In Lei Chen, Peter

Triantafillou, and Torsten Suel, editors, *WISE*, volume 6488 of *Lecture Notes in Computer Science*, pages 262–270. Springer, December 2010.

[HAA11] Roxana Horincar, Bernd Amann, and Thierry Artières. Online Refresh Strategies for RSS Feed Crawlers. In *BDA'11, 27èmes journées Bases de Données Avancées*, Rabat, Maroc, October 2011.

[HAA12] Roxana Horincar, Bernd Amann, and Thierry Artières. Online Change Estimation Models for Dynamic Web Resources. In *ICWE'12, The 12th International Conference on Web Engineering (ICWE)*, Berlin, Germany, July 2012.

[HD04] Younes Hafri and Chabane Djeraba. High performance crawling system. In Michael S. Lew, Nicu Sebe, and Chabane Djeraba, editors, *Multimedia Information Retrieval*, pages 299–306. ACM, 2004.

[HVT+11] Z. Hmedeh, N. Vouzoukidou, N. Travers, V. Christophides, C. du Mouza, and M. Scholl. Characterizing Web Syndication Behavior and Content. In *WISE'11, The 11th International Conference on Web Information System Engineering*, LNCS, pages 29–42, Sidney, Australia, October 2011.

[inta] Internet Archive. www.archive.org/.

[intb] Internet Memory Foundation. http://internetmemory.org/en/.

[JA06] Seung Jun and Mustaque Ahamad. FeedEx: collaborative exchange of news feeds. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *WWW*, pages 113–122. ACM, 2006.

[LAT07] Renaud Lambiotte, Marcel Ausloos, and Mike Thelwall. Word statistics in Blogs and RSS feeds: Towards empirical universal evidence. *CoRR*, abs/0707.2191, 2007.

[LRS05] Hongzhou Liu, Venugopalan Ramasubramanian, and Emin Gün Sirer. Client Behavior and Feed Characteristics of RSS, a Publish-Subscribe System for Web Micronews. In *Internet Measurment Conference*, pages 29–34. USENIX Association, 2005.

[NH01] Marc Najork and Allan Heydon. High-Performance Web Crawling. Technical report, SRC Research Report 173, Compaq Systems Research, 2001.

[nmi] NM Incite. http://www.nmincite.com/.

[NW01] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *WWW*, pages 114–118, 2001.

[OP08] Christopher Olston and Sandeep Pandey. Recrawl scheduling based on information longevity. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *WWW*, pages 437–446. ACM, 2008.

[O'R05] Tim O'Reilly. What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software. *URL http://oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html*, 2005.

[OW02] Chris Olston and Jennifer Widom. Best-effort cache synchronization with source cooperation. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 73–84. ACM, 2002.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[PDO04] Sandeep Pandey, Kedar Dhamdhere, and Christopher Olston. WIC: A General-Purpose Algorithm for Monitoring Web Information Sources. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 360–371. Morgan Kaufmann, 2004.

[pee] Peersim. http://peersim.sourceforge.net/.

[PO05] Sandeep Pandey and Christopher Olston. User-centric Web crawling. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 401–411. ACM, 2005.

[PO08] Sandeep Pandey and Christopher Olston. Crawl ordering by search impact. In Marc Najork, Andrei Z. Broder, and Soumen Chakrabarti, editors, *WSDM*, pages 3–14. ACM, 2008.

[PRC03] Sandeep Pandey, Krithi Ramamritham, and Soumen Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *WWW*, pages 659–668, 2003.

[pub] PubSubHubbub protocol. http://code.google.com/p/pubsubhubbub/.

[RCYT08] Haggai Roitman, David Carmel, and Elad Yom-Tov. Maintaining dynamic channel profiles on the web. *PVLDB*, 1(1):151–162, 2008.

[RMP+07] Ian Rose, Rohan Murty, Peter R. Pietzuch, Jonathan Ledlie, Mema Roussopoulos, and Matt Welsh. Cobra: Content-based Filtering and Aggregation of Blogs and RSS Feeds. In *NSDI*. USENIX, 2007.

[rosa] RoSeS Project. http://www-bd.lip6.fr/roses/doku.php.

[rosb] Architecture d'un Système RoSeS centralisé. http://www-bd.lip6.fr/roses/lib/exe/fetch.php?media=livrables:d1.2_architecture_centralisee.pdf.

[RPS06] Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gün Sirer. Corona: A High Performance Publish-Subscribe System for the World Wide Web. In *NSDI*. USENIX, 2006.

[rssa] RSS Board. http://www.rssboard.org/.

[rssb] RSS Autodiscovery. http://www.rssboard.org/rss-autodiscovery/.

[rssc] RSScache. http://www.rsscache.com/.

[RUM+11] Sandro Reichert, David Urbansky, Klemens Muthmann, Philipp Katz, Matthias Wauer, and Alexander Schill. Feeding the world: a comprehensive dataset and analysis of a real world snapshot of web feeds. In David Taniar, Eric Pardede, Hong-Quang Nguyen, J. Wenny Rahayu, and Ismail Khalil, editors, *iiWAS*, pages 44–51. ACM, 2011.

[Sap06] Gilbert Saporta. *Probabilités, analyse des données et statistique*. Technip, 2006.

[SCC07] Ka Cheung Sia, Junghoo Cho, and Hyun-Kyu Cho. Efficient Monitoring Algorithm for Fast News Alerts. *IEEE Trans. on Knowl. and Data Eng.*, 19(7):950–961, 2007.

[SCH+07] Ka Cheung Sia, Junghoo Cho, Koji Hino, Yun Chi, Shenghuo Zhu, and Bell L. Tseng. Monitoring RSS Feeds Based on User Browsing Pattern. In *Proceedings of the International Conference on Weblogs and Social Media (Boulder Colorado, March 2007)*, pages 161–168, 2007.

[sit] Sitemaps. http://www.sitemaps.org/.

[SMPD05] Daniel Sandler, Alan Mislove, Ansley Post, and Peter Druschel. FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification. In Miguel Castro and Robbert van Renesse, editors, *IPTPS*, volume 3640 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 2005.

[SS02] Vladislav Shkapenyuk and Torsten Suel. Design and Implementation of a High-Performance Distributed Web Crawler. In Rakesh Agrawal and Klaus R. Dittrich, editors, *ICDE*, pages 357–368. IEEE Computer Society, 2002.

[SS09] Uri Schonfeld and Narayanan Shivakumar. Sitemaps: above and beyond the crawl of duty. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *WWW*, pages 991–1000. ACM, 2009.

[STCR10] Adam Silberstein, Jeff Terrace, Brian F. Cooper, and Raghu Ramakrishnan. Feeding frenzy: selectively materializing users' event feeds. In Ahmed K. Elmagarmid and Divyakant Agrawal, editors, *SIGMOD Conference*, pages 831–842. ACM, 2010.

[Ste91] James Stewart. *Calculus: Early Transcendentals*. Brooks/Cole, 1991.

[syn] Syndic8. http://www.syndic8.com/.

[TATV11] Jordi Creus Tomàs, Bernd Amann, Nicolas Travers, and Dan Vodislav. RoSeS: A Continuous Content-Based Query Engine for RSS Feeds. In Ab-

delkader Hameurlain, Stephen W. Liddle, Klaus-Dieter Schewe, and Xiaofang Zhou, editors, *DEXA (2)*, volume 6861 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2011.

[tec] Technorati. http://technorati.com/.

[tim] Time's Person of the Year: You. http://www.time.com/time/magazine/article/0,9171,1570810,00.html.

[TPF06] Mike Thelwall, Rudy Prabowo, and Ruth Fairclough. Are raw RSS feeds suitable for broad issue scanning? A science concern case study. *JASIST*, 57(12):1644–1654, 2006.

[TV00] Vittorio Trecordi and Giacomo Verticale. An Architecture for Effective Push/Pull Web Surfing. In *ICC (2)*, pages 1159–1163, 2000.

[twi] Twitter. http://twitter.com/.

[URM+11] David Urbansky, Sandro Reichert, Klemens Muthmann, Daniel Schuster, and Alexander Schill. An Optimized Web Feed Aggregation Approach for Generic Feed Types. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[wika] Wikipedia. http://www.wikipedia.org/.

[wikb] Wikipedia edit rates. http://www.wikichecker.com/editrate/.

[WSY+02] Joel L. Wolf, Mark S. Squillante, Philip S. Yu, Jay Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *WWW*, pages 136–147, 2002.

[XI05] Rui Xu and Donald C. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[yah] Yahoo! Groups. http://groups.yahoo.com/.

[you] YouTube. http://www.youtube.com/.

[ZSA09] Yongluan Zhou, Ali Salehi, and Karl Aberer. Scalable Delivery of Stream Query Results. *PVLDB*, 2(1):49–60, 2009.

[ZTB+07] Christian Zimmer, Christos Tryfonopoulos, Klaus Berberich, Gerhard Weikum, and Manolis Koubarakis. Node Behavior Prediction for Large-Scale Approximate Information Filtering. *1st International Workshop on Large Scale Distributed Systems for Information Retrieval (LSDS-IR 2007)*, 2007.

[ZTB+08] Christian Zimmer, Christos Tryfonopoulos, Klaus Berberich, Manolis Koubarakis, and Gerhard Weikum. Approximate Information Filtering in Peer-to-Peer Networks. In James Bailey, David Maier, Klaus-Dieter Schewe,

Bernhard Thalheim, and Xiaoyang Sean Wang, editors, *WISE*, volume 5175 of *Lecture Notes in Computer Science*, pages 6–19. Springer, 2008.