Monitoring RSS Feeds

George Adam
(Research Academic Computer Technology Institute and
Computer Engineer and Informatics Department, University of Patras, Greece
adam@cti.gr)

Christos Bouras
(Research Academic Computer Technology Institute and
Computer Engineer and Informatics Department, University of Patras, Greece
bouras@cti.gr)

Vassilis Poulopoulos
(Research Academic Computer Technology Institute and
Computer Engineer and Informatics Department, University of Patras, Greece
poulop@cti.gr)

Abstract: The expansion of the World Wide Web has led to a chaotic state where the users of the internet have to face and overcome the major problem of discovering information. For the solution of this problem, many mechanisms were created based on crawlers who are browsing the www and downloading pages. In this paper we describe "advaRSS" crawling mechanism which intends to be the base utility for systems offering collections of news in real time to internet user. In contrast to the common crawling mechanisms our system is focused on fetching the latest news from the major and minor portals worldwide by utilizing their RSS feeds. The news is produced in a random order any time of the day and thus the freshness of the offline collection can be measured even in minutes. This means that the system has to be updated with news every single time they occur. In order to achieve this we utilize the communication channels that exist on the modern architecture of the WWW and more specifically in the architecture of Web 2.0. As the RSS feeds are used by every major and minor portal it is possible to keep our crawler up to date and retain a high freshness of the "offline content" that is maintained in our system's database.

Keywords: focused crawler, RSS, feed monitoring
Categories: H.3.3, I.7.1

1 Introduction

Web crawlers are an essential component of all search engines and are increasingly becoming important in data mining and other indexing applications. They are programs which browse the Web in a methodical, automated manner and are used to create a copy of all the visited pages for future use by mechanisms which will index the downloaded pages to provide fast searches and further processing.

Maintaining web archives requires an incremental crawler which is capable of revisiting known URLs independently and adaptively to their estimated rate of publishing new content. The goal is to keep the local repository as up to date as possible. In [Brewington, 00a] and [Brewington, 00b] is denoted that most web pages

in the US are modified during the US working hours a statement that is extremely logical and helpful. In [Cho, 99], Cho and Garcia-Molina show that different domains have very different "page change" rates. An approach that utilizes this fact is a scheduling mechanism which estimates the next update timing of web pages based on their update history using the Poisson process [Tamura, 08]. In [Souza, 07] is proposed an additional feature, which includes the politeness constraint which indicates that we may only probe the source at most n times and that no two probes may be spaced less than delta time units apart. This policy is intended to minimize the required bandwidth and to prevent the crawler from being blocked from the source. Finally, in [Sia, 07] Sia et al. study how the RSS aggregation services should monitor the data sources to retrieve new content quickly using minimal resources and to provide its subscribers with fast news alerts. Their experiments prove that, with proper resource allocation and scheduling, an "RSS aggregator" can provide useful content significantly fast.

Apart from the freshness other issues also occur when creating a distributed crawler, either with use of terminals or multi-threaded. The distribution of resources among the crawlers and the communication between them is a matter of discussion. In [Najork, 01], [Wolf, 02] and [Cho, 98] some specific strategies are introduced for effective crawling and parallel crawling. The basic idea that lies behind parallel crawling is a manager which is assigned with the task of organizing the set of terminals-crawlers that access and download pages. In some cases, the terminals crawlers maintain their own database. This implies that the manager is assigned with the task of getting data from the databases and committing the changes to the central database. Additionally, the manager should read and update accordingly the distributed databases with data collected from every terminal-crawler in order to prevent situations of duplicate entries.

In this paper we describe advaRSS, a crawling mechanism that utilizes the communication channels that exist on the architecture of Web which is nothing more than RSS feeds. The idea is the same as a usual crawler with the starting feed URL being the RSS feed (XML file) and the depth of search set to 1, which means follow only the links found in the first page (feed URL). In contrast to the common crawling mechanisms our system is focused on fetching only news articles from the major and minor portals worldwide (multilingual), in order to deliver personalized content to users. The news is produced in a random order any time of the day and thus the crawling mechanism must periodically poll the sources and check for changes many times per day. To make this resource intensive task more efficient, the system has to learn to predict the rate that an RSS is publishing articles, based both on the static features and the complete posting history.

The rest of the paper is structured as follows. In the following section we describe the architecture of advaRSS while in the third section we display the flow of information. In the fourth section, we discuss the algorithmic aspects of the system and then we describe the experimental procedures that were conducted in order to evaluate the crawling mechanism. This paper finishes with some discussion on the mechanism and we conclude with remarks and future work on the system.

2 Architecture

The architecture of the mechanism consists of multiple sub-systems which are assigned with specific roles in order to achieve scalability. The basic parts of the system are (a) the centralized database, (b) the crawler's controller and (c) the terminals that execute the fetching and analysis.

The database is used for storing permanent information such as links to RSS and news articles. Each RSS stored in the database is followed by 24 indicators, one for each hour, which changes dynamically according to the rate that the RSS publishes new articles. Additionally, the database stores information concerning the articles that are fetched utilizing the RSS feeds like the title, the HTML code, the URL, the language, the category and any other information that can be useful by the mechanism that uses this data.

The controller is used in order to organize the whole procedure and is assigned with two major tasks. The first is the direct communication with the database (only the controller can interact with the database) and the second is the job assignment and checking of the terminals. The controller is the part of the mechanism that includes the main procedures and feeds the terminals with URLs from which to download information. In parallel, the controller examines the outputs of the terminals' analysis and stores any information to the database.

The mechanism is utilizing a central MySQL[1] database for permanent storage of data. The system runs every few minutes and the controller is responsible to specify which RSS feeds have to be revisited. This decision is made by estimating the number of new articles that are expected to be published from the last time that the RSS was retrieved. In order to estimate this number, the controller utilizes the hourly posting rates that are stored in the database and the last retrieval time of each source.

Finally, the terminals are used in order to accelerate the crawling procedure. They fetch and analyze the RSS files that the controller has indicated and download every new article that will be found. Thus, the controller will not have to retrieve any RSS feed and communicates only with the terminals that are supposed to send the new articles without any delay. This parallel architecture utilizes the network connections of the terminals to broaden the overall bandwidth of the mechanism. This means that the resource constraints are lowered and the performance of the crawler is increased.

[1] http://www.mysql.com – MySQL DBMS

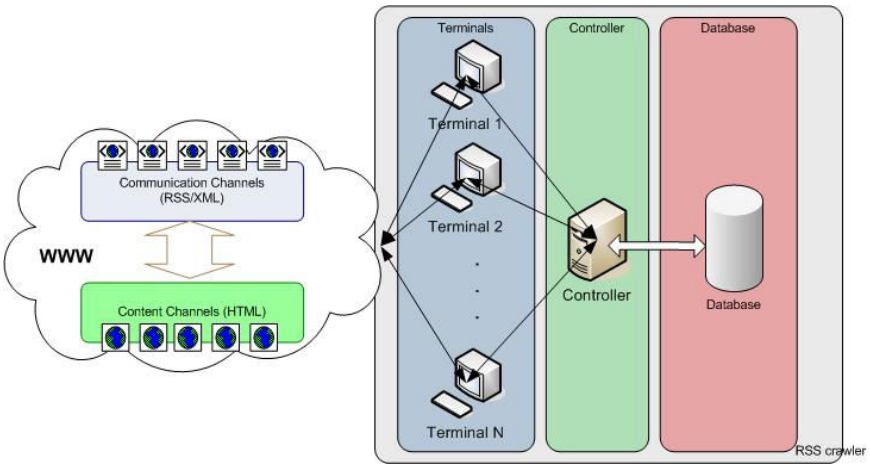

Figure 1: architecture of advaRSS

3 Flow of Information

The main procedure of the mechanism can be decomposed into multiple simple tasks. Firstly, the controller gets a list with all the available RSS feeds that are stored in the database. The list will be filtered in order to process only a small amount of feeds each time. The filtering is done by applying the politeness constraint and estimating the expected new articles from each feed. Afterwards, every RSS feed from the aforementioned list will be processed by an available terminal or by the terminal with the lower resource consumption.

The terminals have a local database in which they keep information about the RSS feeds which is their MD5 hash, the last articles that they provided and their last retrieval time. Thus, when a terminal receives a URL to parse, it requests from the corresponding web server to receive the XML file, only if it has been modified since the last date that was fetched. When the web server sends the file, the terminal checks its hash code compared to the hash code that exists to the local database. We consider that the same hash code indicates an unchanged XML file and thus an RSS that is not updated with new articles.

A retrieved RSS file is examined using an XML parser by a terminal, in order to extract the titles and URLs of the articles that are located into the feed. The terminal checks if any of these titles can be found in its local database. We consider that multiple same titles can be found across different RSS feeds but if the same title has been provided by the same RSS at the previous retrieval, then we consider that we already have the article in our repository. For every article that its title does not exist in the database of the terminal, we fetch and send to the controller the following information: title, url, date (of fetching), html and rss identifier from which it is fetched. Finally, the controller stores this information to the central database and updates the posting rate of the RSS that was examined. The tasks are depicted in figure 2.

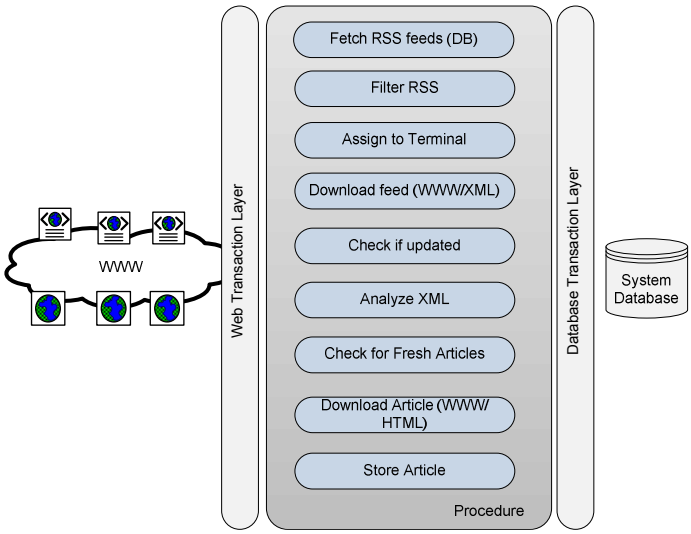

Figure 2: flow of information

4 Algorithm description

As already mentioned in the previous paragraphs we intend to create a crawler that will be able to receive the latest updates of news articles of communication channels and store the information into a centralized database so as to be used from every mechanism that supports presentation of news to Internet users. For every RSS in the database, the system maintains an hourly posting rate history and the last retrieval time. The history is used in order to present the posting activity of the source and to predict the rate that the RSS is publishing new articles, based on the fact that an RSS tends to have similar posting rate at a specific hour, each day.

One of the most basic parts of the system is the scheduling mechanism. It is a subsystem of the crawler that, by using the aforementioned posting pattern, manages to schedule the next visit to the RSS feed. During each subsequently visit, the historic information is updated and new predictions are made, leading to a system that is able to adapt on the RSS updating behavior. We concluded to this algorithm by observing how new articles are published by an RSS during the 24 hours of a day. The following diagram shows the average number of articles posted per hour, for a random RSS in our database.

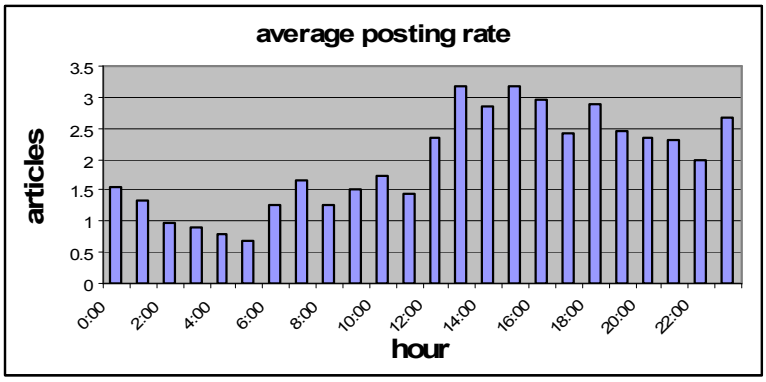

Figure 3: average number of new published articles per hour, for a specific RSS

As it is obvious from Figure 3, the mechanism should schedule more frequent visits at hours with high posting rate. Having the hourly past posting pattern of an RSS and the last time that it was retrieved, we can use the following equation to estimate the expected number of new posted articles since the last retrieval, using the precision of 1 second:

$$articles(t_{now}) = \int_{last}^{t_{now}} \frac{posting\_rate(t_{now} - t)}{3600} dt \qquad (1)$$

The above equation utilizes posting rate per second by dividing the hourly rate by the total number of seconds in one hour. It is obvious that we expect a high number of new articles from an RSS feed with high posting rate. Due to resource constraints, the mechanism is able to perform only limited retrievals per time period. Thus, the crawler has to decide which RSS sources to contact in order to fetch as many articles as it can. A simple monitoring algorithm that utilizes the Eq. 1 to schedule a total of k retrievals for each execution should estimate the expected new articles and select the first k with the higher number of articles. Utilizing the fact that the advaRSS crawler can be used as a part of a system offering collections of news in real time to internet users, we can increase its efficiency by including information about the users. The number of subscribers of each feed and their activity on the system can be used in order to modify the above monitoring algorithm. We assume that if a source has more subscribers than others, it should be retrieved with higher priority, in case that all of them have similar posting rates. Putting the above together on a unique ranking metric, we have for an RSS feed f:

$$rank(f,t) = articles_f(t) \cdot (1 + c \cdot subscribers(f)) \qquad (2)$$

The parameter c in the above metric can be adjusted to reflect how important the information about the number of subscribers is. In systems that the number of subscribers for each source is unknown, this constant must be set to zero, which means that the ranking metric will not use this information at all. Additionally, the scheduling mechanism that we present, takes into account the politeness constraint, which means that no two subsequent retrievals may be performed in less than x time units apart. We assume that a user can tolerate a delay of 10 minutes for an article,

thus we set the minimum time between two subsequent retrievals to this time period. Finally, the mechanism is be able to update the posting pattern for each RSS based on the result of the next retrieval, which is the number of new articles that have been published.

Algorithm 1. Monitoring algorithm

```
feeds[] = Database_Query();
feeds[] = Apply_Politeness_Constraint(feeds[]);
Foreach(feeds[] as feed)
  ranks[feed] = rank(feed, now);
End For
feeds_sorted[] = descending_sort(ranks[]);
To_be_retrieved[] = feeds_sorted[1..k];

Foreach(To_be_retrieved[] as current_feed)
  Assign to terminal {
    XML_Code = fetch_Data(current_feed);
    Extracted_articles[] = Analyze(XML_Code);
    new_articles = find_new_articles(Extracted_articles[]);
  }
  update_DB(current_feed, new_articles);
End For
```

The system is initialized using an hourly posting rate that equals to a constant value for each RSS feed. This assures that all feeds will be treated equally for the first day. The updating process of the posting rates, distributes the number of new articles to each hourly rate, making the mechanism able to adapt to each source. However, a source may have not published a new article for a day, which means that it's posting pattern and ranking metric will be equal to zero. Thus the above algorithm will not retrieve this source ever again. To overcome this problem, the mechanism uses a minimum value greater than zero for the posting rates.

5 Experimental evaluation

In this section, we compare the proposed monitoring algorithm to other retrieval policies. The experiment procedure lasted 90 days and was conducted using RSS feeds from major and minor portals and weblogs.

At the first experiment we put focus on the maximum number of pending articles that a source can have. As pending articles we define the articles that are published but have not been retrieved. The comparison is made using other two monitoring policies. The first is a round-robin policy, which places the RSS feeds in a queue and schedule the retrievals using the FIFO method which means that a source will be revisited after all others have been processed. The second policy uses the posting pattern in order to minimize the total delay of the fetched articles. The delay is defined as the time period between the publishing and the retrieval time.

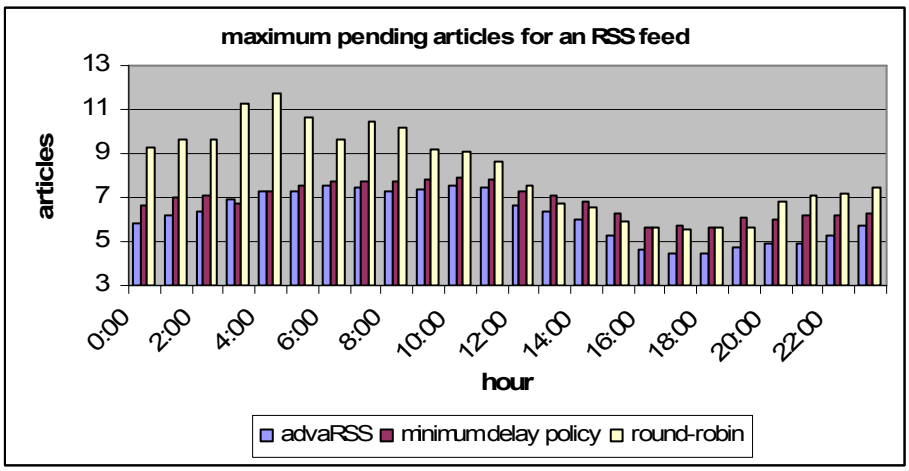

Figure 4: average number of articles, of the RSS with the maximum pending articles.

As it is obvious from Figure 4, the policy that minimizes the total delay increases by 11.2% the maximum pending articles on the source. Using the round-robin policy we notice an increment of 33.4%. The number of articles is an average of the daily measurements. Apart from the above metric, it is interesting to estimate the total pending articles on the system. Thus, the second experiment was conducted using 460 sources and the objective was to calculate the summary of the articles that have not been retrieved yet, for all RSS sources. Both experiments were made by collecting information about these feeds for a period of three months and applying the aforementioned policies.

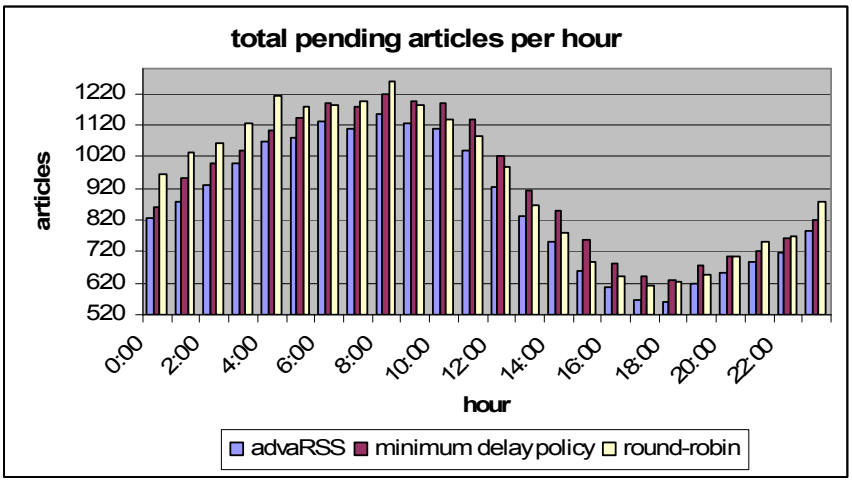

Figure 5: total pending articles on system, per hour.

Figure 5 shows that the pending articles are depending on the posting rate for each hour. We can see that with the "minimum-delay" policy, the total number of pending articles is 7.5% more than the proposed policy. Finally, the result of the round-robin policy is approximately 8.5% more articles. For the experiments, the average retrieval rate is 15% of total feeds per hour.

6 Conclusions

In this paper, we described the architecture and implementation details of our crawling system and presented some preliminary experiments. We highlighted the importance of utilizing RSS feeds in order to retrieve useful content from the Web and how this can be efficiently implemented on a system that has limited resources.

In our mechanism the focus is put on the adaptation of the mechanism to each RSS source, as it is obvious that different sources are publishing new content with different rates. In a World Wide Web that has grown enough from the time of its invention, the personalization issue seems to make the difference, and seems to be one of the most important of our era. The advaRSS intends to be the base utility for systems offering collections of news in real time to internet user such as peRSSonal[2] [Bouras, 07], [Bouras, 08] which is a single web place that offers, in a unified way, personalized and dynamically created views of news deriving from RSS feeds.

7 Future Work

There are obviously many improvements to the system that can be made. A major open issue for future work is a detailed study of how the system could become even more distributed to minimize the resource constraints, retaining though quality of the content of the crawled pages. When a system is distributed, it is possible to use only one of its components or easily add a new one to it. Additionally what we have to do is to compare the results of our crawler with more implementations of other incremental crawlers that selectively chose which pages to crawl.

References

[Bouras, 07] Bouras, C., Poulopoulos, V., Tsogkas, V.: Efficient Summarization Based On Categorized Keywords. The 2007 International Conference on Data Mining (DMIN07), Las Vegas, Nevada, USA. 25 - 28 June 2007.

[Bouras, 08].Bouras, C., Poulopoulos, V. Tsogkas, V.: PeRSSonal's core functionality evaluation: Enhancing text labeling through personalized summaries. Data and Knowledge Engineering Journal, Elsevier Science, 2008, Vol. 64, Issue 1, pp. 330 – 345.

[Brewington, 00a] Brewington, B. E., Cybenko, G.: How dynamic is the web? In Proceedings of the 9th World Wide Web Conference (WWW9). January 2000.

[2] http://perssonal.cti.gr/ - News indexing system

[Brewington, 00b] Brewington, B. E., Cybenko, G.: Keeping up with the Changing Web. In Proceedings of the 9th World Wide Web Conference (WWW9). January 2000.

[Cho, 98] Cho, J., Garcia-Molina, H., Page, L.: Efficient Crawling through URL ordering. In Proceedings of 7th World Wide Web Conference (WWW7), 1998.

[Cho, 99] Cho, J., Garcia-Molina, H.: The Evolution of the Web and Implications for an Incremental Crawler. Department of Computer Science, Stanford, December 2, 1999.

[Najork, 01] Najork, M., Wiener, J. L.: Breadth-first search crawling yields high quality pages. In Proceedings of the 10th World Wide Web Conference (WWW10), May 2001.

[Sia, 07].Cheung Sia K., Cho, J., Cho, H.-K.: "Efficient Monitoring Algorithm for Fast News Alerts," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 7, pp. 950-961, July 2007

[Souza, 07].Souza, C., Laber, E., Valentim, C., Cardoso, E.: A Polite Policy for Revisiting Web Pages, la-web,pp.128-135, Latin American Web Conference (LA-WEB 2007), 2007.

[Tamura, 08].Tamura, T., Kitsuregawad, M.: Evaluation of Scheduling Methods of an Incremental Crawler for Large Scale Web Archives, Abstracts of IEICE transactions on Information and Systems (Japanese) Vol.J91-D No.3, pp.551-559, 2008.

[Wolf, 02] Wolf, J., Squillante, M., Yu, P., Sethuraman J., Ozsen, L.: Optimal Crawling strategies for web search engines. In WWW2002, 2002, ACM.