Utilizing RSS feeds for crawling the Web

George Adam
Research Academic Computer Technology Institute
Rion, Patras, Greece, GR26500
adam@ceid.upatras.gr

Christos Bouras, Professor
Research Academic Computer Technology Institute,
Rion, Patras, Greece, GR26500
bouras@cti.gr

Vassilis Poulopoulos, MsC
Research Academic Computer Technology Institute,
Rion, Patras, Greece, GR26500
poulop@cti.gr

***Abstract*— We present "advaRSS" crawling mechanism which is created in order to support peRSSonal, a mechanism used to create personalized RSS feeds. In contrast to the common crawling mechanisms our system is focalized on fetching the latest news from the major and minor portals worldwide by utilizing their communication channels. The challenge between "advaRSS" and a usual crawler is the fact that 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 almost every modern news portal. 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 by applying algorithms in order to observe the temporal behaviour of each RSS feed.**

Keywords-rss crawling, web crawler, rss analysis, offline content.

I. INTRODUCTION

The World Wide Web has grown from a few thousand pages in 1993 to more than three billion pages at present. The consequence of the popularity of the Web as a global information system is that it is flooded with a large amount of data and information and hence finding useful information on the Web is often a tedious and frustrating experience. New tools and techniques are crucial for intelligently searching for useful information on the Web.

However, the mechanisms that were invented to make Web seem less chaotic need information and waste a great amount of time in order to collect it. Web crawlers are an essential component of all search engines and are increasingly becoming important in data mining and other indexing applications. Web crawlers are programs which browse the Web in a methodical, automated manner. They are mainly 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.

Much research has been done for creating crawlers that will have "fresh" collection of web pages. Web pages are changing at different rates which means that the crawler should decide which page should be revisited by using an efficient method [1]. This leads to creation of crawlers that have at least two basic modules, one for periodical crawling (scheduled) and another for incremental crawling (update the most frequently changing pages). In [2] and [3] is denoted that most web pages in the US are modified during the US working hours a statement that is extremely logical. In [4], Cho and Garcia-Molina show that different domains have very different "page change" rates. Arasu et al in [5] report a half-life of 10 days for web pages in order to create an algorithm for maintaining the freshnesh of their "offline collection".

Apart from the freshness other issues also occur when creating a crawler. Especially when creating a distributed crawler, either with terminals or multithreaded, the distribution of resources among the crawlers and the communication between them is a matter of discussion. In [6], [7], [8] and [9] some specific strategies are introduced for effective crawling and for parallel crawling. The basic idea that lies behind parallel crawling is a manager which is organizing the set of terminals-crawlers that access and download pages.

In this paper we describe advaRSS, a crawling mechanism, which is created in order to support "peRSSonal" [10, 11], a mechanism that produces personalized RSS feeds. In contrast to the common crawling mechanisms our system is focalized on fetching only news articles from major and minor portals worldwide. The difference between advaRSS and a usual crawler is the fact that the news is produced in a random order any time of the day and thus the crawling mechanism has to be efficient in order to obtain each news article that occurs in a portal immediately after its publishing. Another difference is that advaRSS intends to feed peRSSonal with information and thus its output should be "easily readable" and accessed. As the mechanism intends to be the base utility for systems offering collections of news in real time to internet user, it has to maintain a fresh collection of the latest news. In order to achieve this we utilize the communication channels that exist on the modern architecture of the Internet. We are referring to the common RSS feeds. The idea is the same as a usual crawler with the starting feed URL being the RSS feed (XML[1] file) and the depth of search set to 1, which means follow only the links found in the first page (feed URL). The difference stands on the fact that our system is furthermore able to adapt its internal procedure to the time that an RSS changes and thus it is possible to learn the temporal behavior of a feed.

---

1 http://www.w3.org/XML/ - Extensible Markup Language

The idea that motivated us in order to create advaRSS crawler is “peRSSonal” mechanism [10, 11]. This mechanism is able to collect, analyze, and represent in a personalized manner news articles deriving from major news portals. PeRSSonal mechanism, which consists of several modules, utilizes among others a module which is crawler written in Java that requires at least 5 minutes for parsing the testing feed URLs (RSS feeds). The challenge was to create a new more efficient crawler that would support peRSSonal.

The remaining of this paper is structured as follows: In the next section we describe the architecture of advaRSS. In section 3 we discuss the algorithmic aspects of the system and in section 4 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 the future work.

II. ARCHITECTURE

The architecture of the mechanism consists of multiple sub-systems which are assigned with specific roles in order to achieve high speeds of between them and between the crawling sub-system and the peRSSonal mechanism. The basic parts of the system are (a) the centralized database (using peRSSonal's database), (b) the crawler's controller and (c) the terminals that execute the fetching and analysis. The database is used for storing permanent information, the controller is used in order to organize and distribute the procedure and finally the terminals are used in order to fetch the HTML pages from the internet.

A single database is used in order to store the location of the RSS feeds, and the outcomes of the parsing procedure. Additionally, the database stores information concerning the articles that are fetched from the advaRSS crawler. It is obvious that the information that is needed about an RSS feed is its URL and some meta-data while for an article we need information like the title, the HTML code, its URL, the language in which it is written, the date that it was fetched and the category (business, entertainment, politics, etc) in which it is pre-classified by the website from which it was fetched.

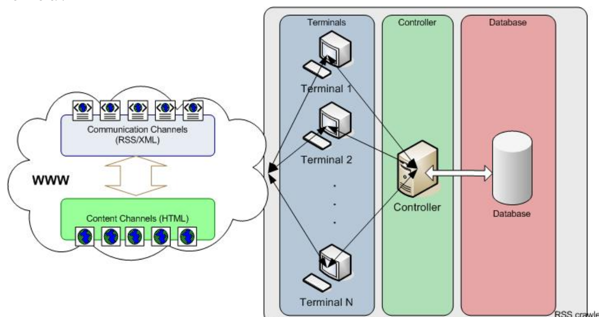

Figure 1: advaRSS architecture

The second sub-system of the RSS crawler mechanism is the controller of the whole procedure. The controller 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. Two kinds of information are usually forwarded for download: (a) URL to XML file and (b) URL to plain HTML file which has to be downloaded. 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 [2]database for permanent storage of data and local databases (files) in order to accelerate the procedure of storing data in the terminal's side. The system runs every 6 to 10 minutes (randomly chosen interval between 360 and 600 seconds) and the algorithmic procedure is as follows:

```
feed_urls[] = Database_Query();
Foreach(feed_urls[] as feed)
    If(Content(feed)==changed) Then
        XML_Code = Fetch_Data(feed);
        Extracted_articles[] =
Analyze_Data(XML_Code);
        For each(Extracted_articles[] as
article)
            If(Not_Exists_in_DB(article)) Then
                Add_to_DB(article);
            End If
        End For
    End If
End For
```

Each RSS stored in the central database is followed by three indicators which are used in order to adapt the frequency that an RSS is examined for new articles and they change dynamically according to the rate of change of each RSS. These three variables are (a) the timer (called T in the rest of the paper), (b) the median (called M in the rest of the paper) and (c) the time of execution (call ToE in the rest of the paper). The most important of these variables is M which is changed according to the temporal behavior of the RSS. It holds a value that indicates how many executions of the mechanism have to be passed between two subsequent examinations of this RSS. T is a counter that is increased M times for every time an RSS occurs to be unchanged and it is used in order to make the calculation of the M variable more efficient. while the ToE variable is a helping countdown variable that is decreased by one whenever the system “runs”. When ToE is equal to zero, the RSS has to be parsed and after the execution ToE is set to be equal to M.

For every RSS that has ToE set to zero, the controller of the system fetches its information from the database and forwards the data to the terminals for execution. The controller stores in its local database information that have to do with the workload of the terminals in order not to overload the terminal with lots of procedures.

[2] http://www.mysql.com – MySQL DBMS

The terminals have also a central local database in which they write and read information that could accelerate the procedure. The actions that could accelerate the procedure are less database transactions and less article retrievals. Thus, the information that is stored to the local databases is the hash code of the last XML file that was fetched concerning an RSS and a file containing all the titles of the RSS that was fetched lately. They also have local database files When a terminal receives a URL to parse, it receives the XML file from the WWW and 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 signal is then sent to the controller that the RSS is unchanged and its T variable is increased by M. If the RSS is changed (different hash code) its T variable is set to 1 and the RSS is analyzed in order to determine the new articles. The titles and URLs of the articles are fetched from the RSS files and the terminal checks to see if any of the title already exists in the local file repository that stores the titles of the RSS that was fetched previously. We consider that multiple same titles can be found across different RSS feeds but only if the title exists in the same RSS we consider that we already "own" the article. The mechanism utilizes the above check in order to reduce the transactions with the central database. For every article that its title does not exist in the previous execution of the mechanism, we fetch and send to the controller the following information: title, url, date (of fetching), html and rss id from which it is fetched. When the controller receives a new article from a terminal, it stores the aforementioned information together with the language of the article and the category of the article, information that derive from the RSS. If no language or category are set to an RSS, English and uncategorized are set to the fields of the database respectively.

III. Algorithmic Aspects

As already mentioned in the previous paragraphs we intend to create a crawler that will be able to receive the latest updates in the 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 three different variables and two files. The variables are used in order to create a system that is able to adapt on the RSS changing behavior. By changing behavior we define the time period which implies changes to an RSS (added articles). The files are used in order to quickly check if an RSS had changed since the last time it was parsed. The variables are stored centrally to a database while the files for every RSS are stored to the terminals. The first file includes the hash code of the XML file that was fetched in the last execution while the second file includes all the titles of the articles that were added in the latest fetched XML file (RSS feed). The hash code is sensitive to minor changes to the XML file and thus we can easily obtain information if it was changed. Even if the hash code differs we are not sure whether there are new articles in the RSS. In this case, we use the second file that includes all the titles of the articles contained in the last fetched RSS and help us realize if the new RSS file that is fetched includes any new articles. This prevents communicating with the database in order to check for any minor or major RSS change.

```
Every X min {
  feeds_to_parse[] = Select RSS feeds from
database with ToE equal to zero;
  Foreach(feeds_to_parse[] as url)
    xml_code = fetch_rss(url);
    If(not modified)
      continue;
    else
      articles_in_rss[] =
extract_info(xml_code);
      Foreach(articles_in_rss[] as article)

If(title_not_found_in_last_articles(article))
          add_to_DB(article);
        End If
      End For
    End If
  End For
}
```

Where:
feeds_to_parse : the array of URLs (rss feeds) to be parsed.
fetch_rss() : returns the XML code of the given URL
last_articles: the titles within an RSS parsed from the last execution of the RSS
This algorithm is utilized in order to observe which RSSs are unchanged since their last parsing by the system and in parallel searches which articles within an RSS were fetched. Additionally, an algorithm is utilized in order to update the execution time of the parsing procedure for an RSS in order not to check every RSS in every execution of the system. This algorithm is applied as the peRSSonal system that is supported by the crawler could have hundreds of RSSs to be parsed and it is not possible to check every single RSS every ten minutes that is the time limit of the system's execution.

```
feeds[]=Fetch_rss_having_zero_ToE();
 Foreach(feeds[] as url)
  If(not modified)
    newTimer=T+M;
    M = M + 30%T
    T = newTimer
  Else
    T=1
    M = 20%M+80%T
  End If
 ToE = M
End For
```

where
M (median): a number represent every how many times an RSS has to be parsed.
T (timer): a number representing how many executions, at least, have passed since the RSS was last changed.
ToE : a counter that is reduced in every execution of the system. When zero, the RSS has to be parsed.

This algorithm is utilized in order to change the ToE of each RSS according to its rate of change which is represented by the timer (T). The median (M) variable is used in order to store the rate of change, which is a value that changes according to the rate of change of an RSS and can learn the behavior of an article.
One of the most basic parts of the system is the execution time updater. It is a subsystem of the mechanism that is able to change the execution time of each RSS which defines every how much minutes an RSS has to be parsed. We concluded to this algorithm by observing how an RSS changed during the 24 hours of a day. The following diagram shows the number of the articles published in an RSS divided by the overall number of articles published by the specific RSS per hour.

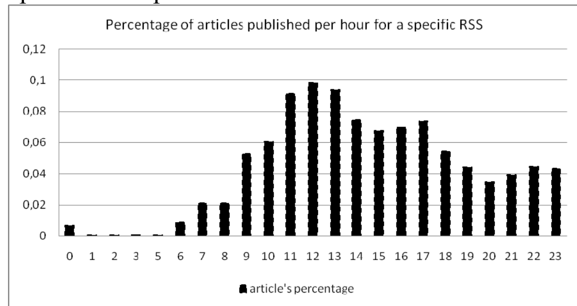

Figure 2: Percentage of articles published by a specific RSS per Hour

As it is obvious from the above figure the system should check the specific RSS very frequently between 11 and 14 o'clock as almost 30% of the articles of the specific RSS are published during these hours while advaRSS should not check very frequently this RSS from 1 to 6 o'clock as less than 2% of the articles are published during these hours. We should find thus a way in order to change the time an RSS is processed by the system. We define the ToE, M and T variables already mentioned in the previous paragraphs. We follow the next equations in order to change their values.
If ToE > 0

$$ToE = ToE \quad 1 \qquad (1)$$

If ToE = 0, there are two different cases which are: (a) no new articles occur in the RSS feed compared to the latest time the RSS was parsed and (b) new articles (one or more) occur.
In the first case we apply the next equations in the presented order:

$$\begin{gathered} Temp = T + M \\ M = M + 30\%T \\ T = Temp \end{gathered} \qquad (2)$$

In the second case we apply the following equations:

$$\begin{gathered} T = 1 \\ M = 20\%M + 80\%T = 20\%M + 0.8 \text{ (because } T=1) \end{gathered} \qquad (3)$$

After each execution (ToE = 0) we set the value of M to ToE:

$$ToE = ceil[M] \qquad (4)$$

From Eq.1 it is obvious that for every execution of the system the ToE is decreased in order to reach the value 0, which indicates that the RSS has to be parsed. When the ToE is zero, then we check the current RSS feed for changes. If no changes are observed then the T variable is increased by M (the times that the RSS was unchanged) and the M variable is changed according to Eq.3. If the file has not changed (no fresh articles) then M increases slightly, while when the file has changed (indicating fresh articles) M decreases dramatically. We concluded to these changes to the Median (M variable) in order to achieve two basic goals: (a) when an RSS is not changing it is checked fewer times per day and (b) when an RSS starts to be updated the Median variable is rapidly decreasing and we manage to adapt the ToE to the rate of change of the article RSS feed. The maximum value for M depends on the frequency of change of the RSS obtained during the first days of the crawler's execution. Starting from a maximum value of 25 the median can be increased in order to reach the value 640. Trying to explain the values, if we check the RSSs every X minutes then the RSS will be checked at least every 25*X minutes which means 58/X times per day in the worst occasion. For a typical value of X set to 8 minutes, an RSS with median set to 25 is checked 7 times per day. This seems to be quite a lot for RSSs that change once a week but still, we have to maintain a fresh collection. On the other hand, the minimum value for M could be 1 indicating check for fresh articles every X minutes.
The initial values of M and T are 4 and 1 consequently. ToE is set with an integer random value between 0 and 3 for each RSS in the database in order to distribute the initialization tasks to 3 executions. This is translated to: when the system begins its first execution, it checks every RSS with ToE equal to zero will be processed. The RSSs will be found to be changed and thus the median will be set directly to 2,4 (M=20%M+80%T). While all the ToEs of the RSS will be already reduced the RSS the ToEs of the RSSs that were just checked will have ToE set to 3 (ceil[ToE] where ToE is set to 2.4). By doing this we assure that the first time that the RSSs are going to be parsed they will be checked at least one time in subsets and after that they will start to adapt on their rate of change.
The basic part of the algorithm is eq. 2 and eq. 3 which update the M variable that represents the rate of change of an article within a specific limit. As the rate of change of an RSS may vary during the year we do not maintain a history of M and we do not base our system on a specific pattern.

IV. EXPERIMENTAL EVALUATION

A crawler has to be adaptive on each URL that it is searching and the workload has to be distributed in order to access parallel a huge amount of data. In that means we have conducted experiments in order to observe the different possible solutions for our crawler and select the most suitable for our occasion. It is expected that a crawler that is processing multiple RSS at the same time (parallelism) will be faster than a crawler that is accessing its feed URL in a serial manner, though we have to observe if the adaptation algorithmic procedure (file checking for duplicate entries and RSS changes) consumes too much time. We conducted an experiment with three different systems and measured the execution time for each of them. The first system was without any adaptation (files) and run on a single computer. The second system distributed the procedure over multiple computers, without any adaptation while the third system was utilizing the adaptation algorithm with multiple terminals.

From the following figure it is obvious that the time of execution by utilizing files for duplicate entries or RSS changes with distributed procedure is the most suitable for our system as it takes only 10 seconds to search for new articles in the RSSs and add the new articles in the database for every time the crawler is running. The time measured is the average time needed for each time the system is executed on the 793 RSS feeds in order to analyze them and fetch all the new articles that occur. In average the system is able to add more than 2500 articles daily to its database.

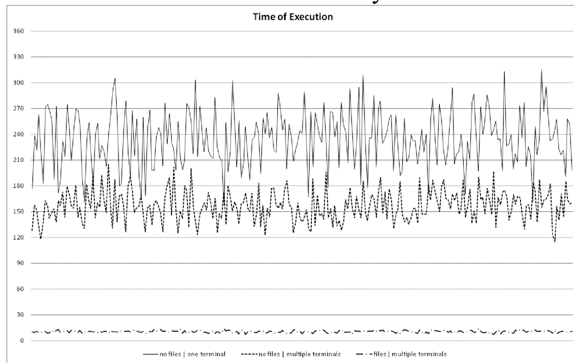

Figure 3: Time of execution of the system under different conditions of system set up

The second adaptation of the system is on the rate of change of each RSS. It is expected that thousands of RSS URLs will be added on the database when in production. This means that the system will have to check every ten minutes, thousands of URLs for new articles (hundreds of RSS X >10 URLs in each RSS feed) every time the crawler is executed. By adapting on each RSS feed rate of change we are able to access the RSS feeds not every time that the crawler is running but every time that the system "believes" that there may be a change to the RSS's content. The algorithm utilized is analyzed in "Algorithmic Analysis" and some experiments follow.

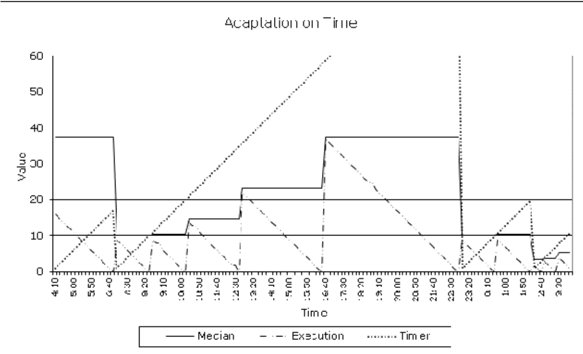

Figure 4: Adaptation of the variable M, ToE and T on Time (1)

As it is obvious from the experiments an RSS is not checked every six to ten minutes that the crawler is executed but it is checked every time that the system "believes" that a change may have occurred. By doing this adaptation we are able to reduce the period of checking an RSS by 5 times. If we were checking each RSS every ten minutes this means that we would need to check an RSS more than 140 times per day. If we think that there are RSSs that do not change daily, then we would consume a lot of resources without any need.

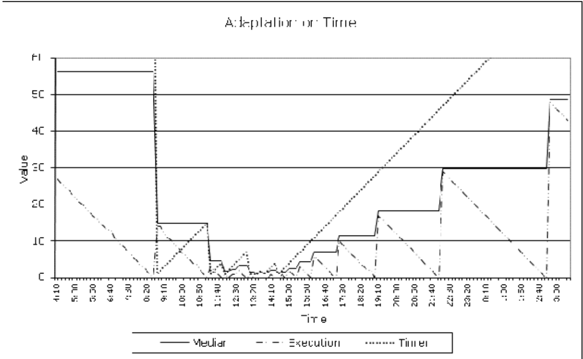

Figure 5: Adaptation of the variable M, ToE and T on Time (2)

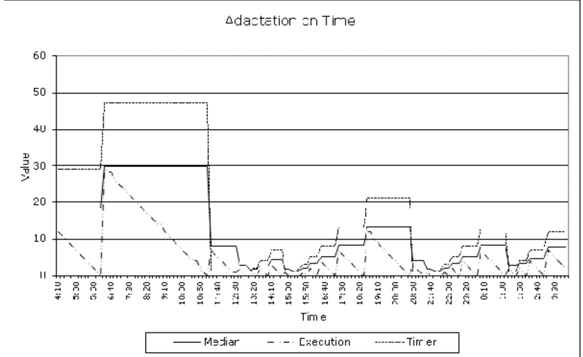

Figure 6: Adaptation of the variable M, ToE and T on Time (3)

By doing the adaptation we check an RSS according to the learning algorithm for the system, which is in average 20 times per day for the RSSs that is updated very often in a day and less than 10 times per day for RSSs that do not change frequently.

An issue that arises from the previous adaptation is how fresh the articles are when they are captured by the RSS

crawler. This could be translated into a simple assumption: if we are not checking the RSSs periodically with a suitable rate, then there is a possibility to add an article to the database with important delay. As long as we want to provide a real time service to the end users, this means that we must have a new article added to the database within a time limit. The time limit that we would like to achieve is at most 30 minutes and we expect this delay to occur under circumstances of delay in update of the RSS feed. In order to be clear, if an RSS is not updated for a short period of time (3-4 hours) then the Median is set to values that indicate searching an RSS every 30-40 minutes. This means that if a delay occurs at this period the system could possible observe the change with a delay of 30-40 minutes.

V. CONCLUSION AND FUTURE WORK

Due to the dynamism of the Web, crawlers form the back-bone of applications that facilitate Web information retrieval. In this paper, we described the architecture and implementation details of our crawling system and presented some preliminary experiments. We explained the importance of extracting only content from web pages and how this can be implemented by a mechanism content analysis, corresponding crawling policies and clever systems that extract content of high quality.

In our mechanism the focus is put on the adaptation of the mechanism to each domain. We showed the importance of adaptation on each domain as it is obvious that the web pages of different domains have different behavior (change in a different manner). 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. We tried to cover this issue through our implementation and try to go a step further by implementing both adaptation on each domain and a selective incremental part. 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, 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 the implementations of other incremental crawlers that selectively chose which pages to crawl. Though, we believe that our system consists of something more than just a crawler. Our intention is to create a clever system that would be able to collect “fresh” content from the web in order to support, with data, mechanisms specialized on data mining, information extraction and categorization.

REFERENCES

[1] H. Bullot, S. K. Gupta and M. K. Mohania. A Data-Mining Approach for Optimizing Performance of an Incremental Crawler. In Proceedings of the IEEE/WIC International Conference on Web Intelligence (2003).

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

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

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

[5] A. Arasu, J. Cho, H. Garcia-Molina, A Paepcke, and S. Raghavan. Searching the Web. In ACM transactions Internet Technology, 2001.

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

[7] M. Najork and J. L. Wiener. High-Performance web crawling. Technical report, Compaq Systems Research Center, 2001

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

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

[10] C. Bouras, V. Poulopoulos, V. Tsogkas. 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.

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