2011 IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE International Conference on Social Computing

Mapping the Blogosphere — Towards a Universal and Scalable Blog-Crawler

Philipp Berger, Patrick Hennig
Justus Bross, Christoph Meinel
IT-Systems Engineering
Hasso-Plattner Institute
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{philipp.berger, patrick.hennig}@student.hpi.uni-potsdam.de,
{jusuts.bross, christoph.meinel}@hpi.uni-potsdam.de

***Abstract*—The massive adoption of social media has provided new ways for individuals to express their opinions online. The blogosphere, an inherent part of this trend, contains a vast array of information about a variety of topics. Thus, it is a huge think tank that creates an enormous and ever-changing archive of open source intelligence. Modeling and mining this vast pool of data to extract and describe meaningful knowledge in order to leverage (content-related) structures and dynamics of emerging networks within the blogosphere is the higher-level aim of the research presented here. While the concept of our tailor-mode feed-crawler was already discussed in two earlier publications this paper focuses on our approach to extend the earlier feed-crawler to a more universal and highly scalable blog-crawler.**

I. Introduction

Since the end of the 90s, weblogs have evolved to an inherent part of the worldwide cyber culture [1]. In the year 2008, the worldwide number of weblogs has increased to more than 133 million [2]. Compared to around 60 million blogs in the year 2006, this constitutes the increasing importance of weblogs in today's internet society on a global scale [3].

Technically, weblogs are an easy-to-use, web-enabled Content Management System (CMS), in which dated articles ("postings"), as well as comments on these postings, are presented in reverse chronological order [4]. Their potential fields of application are numerous, beginning with personal diaries, reaching over to knowledge and activity management platforms, and finally to enabling content-related and journalistic web offerings [5][6].

One single weblog is embedded into a much bigger picture: a segmented and independent public that dynamically evolves and functions according to its own rules and with ever-changing protagonists, a network also known as the *blogosphere* [7]. A single weblog is embedded into this network through its trackbacks, the usage of hyperlinks as well as its so — called *blogroll* — a blogosphere-internal referencing system.

This huge think tank creates an enormous and ever changing archive of open source intelligence [8]. However, the biggest congeniality of the blogosphere — the absence or independence of any centralized control mechanism — is meanwhile its biggest shortcoming: Modeling and mining the vast pool of data generated by the blogosphere to extract and represent meaningful knowledge seemed so far virtually impossible.

II. Project Scope

Facing this unique challenge we initiated a project we already described in other publications with the objective to map, and ultimately reveal, content-oriented network-related structures of the *blogosphere* by employing an intelligent *feed crawler*. A crawler, also known as an ant, automatic indexer, worm, spider or robot, is a program that browses the World Wide Web (WWW) in an automated, methodical manner [9].

To create relevant results out of this big amount of data it is necessary to analyze it based on different techniques and algorithms. For example the knowledge about atmospheric pictures or trends for a specific topic can be very important for a lot of organizations or a special group of people like politcians. There are three important parts, the crawler described in detail in this paper, an analysis framework and a visualization to make the results comfortable to read for humans. To complete this overall approach we started a project called *Blog-Intelligence*. The current implementation of the project is already working as a prototype for the German *blogosphere*[1].

To allow processing of the enormous amount of content in the *blogosphere*, it is necessary to make that content offline available for further analysis. This task is prototypical implemented by the current feed-crawler. This paper discusses to develop a general and scalable blog-crawler based on the current prototype.

While the next section focuses on the current crawler functionality and its corresponding workflows, section four discusses the new concepts for enhancing the crawler such as storing the data to an in-memory database, using *MapReduce*, extracting posts and calculating an adequate update interval. Based on these insights, we provide in section six an outlook as well as recommendations for further research with the overall objective to further enhance our crawler. Section seven gives a conclusion, followed by the list of references.

[1] http://www.blog-intelligence.com

978-0-7695-4578-3/11 $26.00 © 2011 IEEE
DOI

III. Current State of the Blog Intelligence Framework

The crawler implementation recognizes posts by crawling only feeds, rss and atom, from a host identified as a blog. The identification of a blog works as described in the following. The webpage is scanned for common patterns of diverse blog systems. For instance, a pattern could be a match against the generator tag of a web page like *<meta content='blogger' name='generator'/>*. Afterwards, if a pattern matches, the crawler downloads the first alternate link *rel="alternate"* with the type of a xml page identified by *type="application/rss+xml"* or *type="application/atom+xml"* and takes the referenced feed as the main feed for a blog which should contain all posts. In some cases, you run the risk that feeds only contain a subset of all posts. For instance, the crawler can pick a feed for a specific category. In addition, it is common that published feeds are limited to the latest posts.

Consequently, the crawler is limited on crawling a subset of all available posts. The current crawler implementation is facing two gaps during the post retrieval. The fact that only the latest posts can be found in the feed creates both gaps . Blogs, which offer past and present posts in their feed, are an exception and usual very small. First, the historical gap represents the posts not covered because the crawler is unable to dive deep in the history of a blog. Second, the update-frequency gap contains posts that are published between two crawls. As soon as the blogger publishes more posts as the feed can impound, the old implementation is not able to recognize these *intermediate* posts.

The length of the update interval is mainly caused by the time to read and write to the database. The crawler spends a lot of time in waiting for queries. In particular, selecting the next job can take about several minutes. This bottleneck limits scalability. Although downloading and processing is highly parallelized, all instances have to wait for the database connection.

Another point for enhancement is the analyzing part of the framework. Currently, the analysis consists of several independent scripts. These scripts work partially on the same database as the crawler and partially on another database used by the visualization. Each of the scripts has to be executed manually, which is very time consuming. In addition, few analyses (e.g. removing spam blogs) have to be executed manually, too. The desired enhancement is a more integrated approach: The analysis works on the same database as the crawler and the visualization. This could also enable the visualization to deliver real-time data. Additionally,the *Blog Intelligence* framework should handle upcoming requests for personalized search . To sum up, all these points lead to the necessity to improve the current crawler implementation.

IV. New Concepts

In this section we present concepts for implementing the suggested improvements for the crawler. First, we discuss a new implementation basis and storage solution. Afterwards, we describe a more general approach what the parsing of the crawler is concerned along with a new scoring concept and update mechanism.

A. MapReduce

By introducing the concept of *MapReduce* to our crawler, we decrease the coupling between the different crawl tasks and increase the degree of parallelization. The *MapReduce* programming model is designed for the processing and generation of large data sets. Hereby the model dictates partitioning of the execution code, called job, into a map and a reduce function. These functions enable the underlying execution framework to distribute the program execution among multiple machines. As consequence, MapReduce allows a high degree of parallelism[10].

In the context of blog crawling, the fetching and parsing of terabytes per day would be possible given a sufficiently big cluster.

B. Apache Nutch Framework

An existing *MapReduce* web crawler implementation is the open-source web search engine *Apache Nutch*[2]. *Apache Nutch* provides a transparent alternative to the private global scale search services [11].

It comes with an easily extensible and scalable crawler component. Following the *MapReduce* model, *Apache Nutch* defines four different jobs for crawling: *generator*, *fetcher*, *parser* and *updater*. The *generator job* selects the next *URLs* to fetch from the database. The *fetcher job* asynchronously downloads the selected pages. Afterwards, the *parser job* extracts metadata, links and the actual text content. In addition, the framework offers an extension point to insert new parsing algorithms. Finally, the *updater job* inverts links and calculates scores for parsed web pages. Each job works on a large amount of pages in parallel. During the crawling process *Apache Nutch* stores his data into a distributed file system *HDFS* [3].

Apache Nutch is a *MapReduce* application dedicated for scale-out scenarios. For example, *Google* uses a cluster of countless small machines to crawl the web. Nevertheless, even on *scale-up* scenarios *MapReduce* applications perform as *scale-out-in-a-box* more effective than pure *scale-up* approaches [12]. This enables us to run the crawler on a large cluster of small machines as well as on a large shared-memory server. In this context, the HPI offers a testing platform: the *Future SOC Lab*[4], which provides researchers access to the latest multi/many-core hardware. This enables us to run our crawler implementation on a *scale-up* scenario.

C. Combine Nutch with In-Memory Databases

As previously mentioned, *Apache Nutch* stores into the *HDFS*. *HDFS* is designed to run on commodity hardware. Thereby, it does not make any assumption about the underlying

[2]http://nutch.apache.org/

[3](Hadoop Distributed File System) http://hadoop.apache.org/hdfs/

[4]http://www.hpi.uni-potsdam.de/forschung/future_soc_lab.html

hardware. The current implementation uses a *PostgreSQL*[5] database that is not able to answer the number of queries of the crawler in acceptable time. Even when running the implementation on latest hardware, we are not able to noticeable decrease the query processing time. Although *PostgreSQL* makes the data discoverable and easily queryable by offering a *SQL* query *API*, we have to go with another memory solution. In contrast to *PostgreSQL*, *HDFS* is able to handle and process large data sets using a low-cost cluster. Thus, *HDFS* stores only meta data in the main memory. However, given the latest hardware, *HDFS* is not able to make full usage of the given amount of main memory. Additionally, *HDFS* does not offer a *SQL* as easy discoverable query *API*.

"In business compliance this is about to change as hardware architecture have dramatically evolved in the last years"[13]. That means that multi-core processors and low cost main memory can leverage the time consuming tasks extremely well. Since costs for main memory is very low and access to files in the main memory is much faster, it makes sense to store the operational data only in the main memory. "In-memory is not a totally new concept, it was introduced in the 1980s" [13], but was not focused any longer since main memory was too expensive. Prof. Dr. Hasso Plattner is focusing on this topic in detail in his book *In-Memory Data Management — An Inflection Point* [13] based on enterprise applications *SAP* is developing.

Likewise enterprise applications, the *blog intelligence* framework has similar analysis aspects. In contrast to a usual crawler, searching for key words is less important. Instead, we have to do analysis that is more complex. We also need to analyze the link structure within the blogosphere. Unlike usual search engines, we are focusing on a trend analysis. Thereby, we want to follow discussions about a certain topic throughout the blogosphere. We need to find out where a discussion has begun, which posts are referring to this post and how a discussion evolves. Another important aspect is to calculate a blog ranking. As described in [14], we want to build a ranking, which is not only based on the connectivity used in usual rankings, but also on the importance of the content.

Because of the effective usage of the main memory and the versatile analysis capabilities, we decided to extend *Apache Nutch* to store all collected data in an in-memory database. Related to that is *Apache Gora*[6], which connects *Apache Nutch* to common databases. Therefore, we have to adapt the *Apache Gora* connector to run on the *SAP* in-memory solution *HANA*[7].

D. Adaptive Post Recognition

Besides performance improvements, we want to close the update gaps of the current implementation described in section III. As discussed, both gaps result from the limited size of feeds. Therefore, we developed a concept for an adaptive algorithm. This algorithm identifies posts on web pages using patterns that are automatically extracted from the blog's feed.

[5]http://www.postgresql.org/

[6]http://incubator.apache.org/gora/

[7]http://www.sap.com/platform/in-memory-computing/in-memory-appliance

The crawler first crawls the whole host of a potential blog. It is quite difficult to define whether all pages of one host are covered. Here we assume that the crawl ends as soon as there are no more pages found whose link-path stays inside the current host.

The crawl returns a set of *URLs* and corresponding documents from one host. We define the *start-page* of one host as the shortest *URL* of this host. This is typically the host itself. Given this page, the crawler searches for an *alternate-xml-link*, which defines the feed of the host. If there is no feed url on the *start-page*, the host will not be recognized as a blog. Otherwise, the feed get fetched and parsed. This gives some blog specific properties like *title* and *author*, but typically also the 10 latest posts of a blog. Among other more specific attributes the data of the posts consists of various core attributes like *permalink*, *title*, *author*, *published date* and *content*. These data enables us to find and analyze the specific web page for each post. It is common, to publish an extra page for each post, often referenced by the *permalink URL*. By identifying the position of the attributes of a post in the *HTML DOM* structure, we can generate specific *path* sets for each blog.

A *path* in the *DOM* tree is the ordered list of all parent nodes starting from a specific node. The first challenge is to find the nodes corresponding to the post attributes. Especially the feed content attribute contains *HTML-style* tags as well as the content on the page itself. Therefore, it is necessary to clean both *HTML* representations that they contain only structure-relevant tags. In best case, post attributes contain no styles and need no cleaning. After the clean up, it is possible to search for an attribute's content in the *DOM* tree.

For example, there are typical two *paths* of the *title* attribute in the *HTML* page of a post at *blogger.com*. The first occurrence is in the *title* tag and has the *path /HTML/head/title/::text()*. The second time the title occurred at */html/body/div/div/div/div/div/.../div/div/h3/::text()*. Here two problems of this approach reveal. The first problem is the handling of no exact matches. This occurs in the title tag, which is a combination of the *blog title* and the *post title*. Here we have to extend the *XPath*[8] notation in such a way that we give relative positions in text nodes, like *start*, *middle* and *end*. (eg. text(*start*)). Second, the node names are insufficient as URI for a node because they often have similar structures created by multiple *div* tags. On second sight, the uniqueness of the *path* can be derived by the *class*, *id* and *style* attributes of the different nodes. Consequently, the full second *path* should look like this */html[@class='v2']/body[class='loading']/div[class='content']/.../h3[class='post-title entry-title']/::text()*. By creating *paths* for each attribute on the page, we get a very specific pattern for attributes of one post in its *HTML* page. Given patterns for all posts, we can reduce the patterns to a common denominator by removing changing attributes like *id*, *href* or *alt*.

Because all blog systems are, in an abstract sense, *web*

[8]http://www.w3.org/TR/xpath/

content management systems (WMS). They give documents an unified visual appearance to provide coherence of the style and design. As a result, we can conclude that posts pages in blogs provide a unified structure and appearance. Therefore, patterns extracted by the latest sample of posts will map to the rest of the posts *HTML* pages. This enables us to extract meta information from posts, which are no longer linked from the blog feed, by using the *path* for each attribute.

E. Blog Coverage

Following our adaptive post recognition approach, the crawler is first downloading every page of a specific host. Afterwards, pages are categorized based on the corresponding post entries from the feed. In contrast, not every page of a host is a post page or a blog relevant page. For example, there are embedded shops, user profiles or other static pages. Therefore we like to narrow the crawl down to just blog relevant pages. Furthermore, we need to ensure that the crawler has downloaded enough pages of a blog host. As consequence, we introduced a customized prioritization for fetching pages.

1) Blog Host Coverage: To ensure the effective coverage of a blog host, we need to shrink the set of pages to crawl. Our first approach is to limit the crawl to a certain depth beginning with the first page identified as blog of a host. Thereby, we focus on pages that are the easiest reachable for users of a blog. It is most likely that these pages contain the most important post pages.

On narrowing-down to *breadth-first crawl*, the chance of getting an old post is low. Usually different blog systems offer a so-called *blog archive*. This archive contains almost all posts ever published on the blog sorted by publishing date. A common implementation is to offer a pageable archive, so the user can flip through the history. Since we want to analyze the development of topics over time, we need to handle these historical posts as well as the recent ones. The easiest approach is to prioritize *URLs*, which lead to archive pages, based on special keywords like *"archive"* or *"p"*. Unfortunately, this only applies to known and unmodified blog system versions. In order to give an universal solution, we want to use an inverted *DUST* approach [15].

The idea behind this is to analyze *URLs* and predict for example other *URLs* where archives of posts can be found. A very important part is to crawl the posts not listed on the first web page. To achieve this, it is necessary to get *URLs*, which look like *http://www.spreeblick.com/archive/2*. If we knew that by incrementing the last number we get all old posts of a blog, we get historical posts faster. This is what can be done by using the *DUST* approach [15].

A set of *URLs* from a host is split into a *prefix*, a *suffix* and the *number*. Afterwards, these triples are grouped by this order. If there are more than, for example, three *URLs* with the same *prefix* but different *numbers*, we assume that it is possible to get other content by further incrementing until it reaches an *error page*. Consequently, it is necessary to increment the variable number in the *URL* until the content stops changing.

2) Page Scoring: Facing the enormous number of outgoing links of a page we need to prioritize which link to crawl first. The prioritization, also known as *scoring*, should enable the crawler to reach relevant pages fast. Relevant pages are in general pages necessary for the capturing of the *blogosphere*. On the one hand, it is important to crawl relevant pages for parsing like feeds and post pages of a blog. On the other hand, we need to discover new blogs especially by exploring via the interconnection between blogs via posts. These requirements lead us to the following scoring model: All links to feeds that identified during crawling are rated with the highest score. Within a feed, outgoing links extracted from permalinks would possibly lead the crawler to potential posts, so we rate these with the second highest score. All links from a web page providing a feed give us the most relevant *HTML* links. For example, they could contain blogroll links, post links, trackbacks and other additional concepts of known blog systems. Therefore, they get the third highest score to discover the interconnectedness. Other pages are not that important for the crawler, because these are *HTML* pages and their outgoing links will lead most probably outside the blogosphere. In addition, we combine this *scoring model* with *breadth-first crawl* approach mentioned in section IV-E1. Therefore, we increment the *score* of a page based on the number of *hops* from the first web page of the current host.

F. Refreshing Blog Data

By using our *adaptive post recognition* approach and by optimizing the coverage of post pages, we get able to close the gaps described in section IV-D. Nevertheless, doing a crawl of the blog host and running the *adaptive post recognition* is expected to face comparable high retrieval effort. Therefore, we decided to compute the best retrieval time for a blog feed. This avoids using the *adaptive post recognition* for the retrieval of at least the *update-frequency gap*. This time depends on the publish rate of the blog itself. To prevent parsing *HTML* pages we need to predict when the feed overflows. For a given feed we can calculate the visit interval by $\min(publish\_interval) \times (number\_of\_entries - 1)$. On revisiting the feed we can validate our prediction by checking if we find at least one already crawled post in the newly retrieved feed. Otherwise, we have to run the *adaptive post recognition* approach against the whole blog host to guarantee full coverage.

G. Resulting Crawling Process

After applying these new concepts to *Apache Nutch*, we develop an enhanced version of the original crawling process described in section IV-B.

We extend the *GenerateJob* with the logic for the refreshing process. Thereby, the crawler selects feeds to update next as described in section IV-F. To extract the blog and post specific attributes, we update the *ParserJob*. In addition, we integrate the blog system detection into the *ParserJob*. To implement the post recognition described in section IV-D, we have to add a new job. This job analyses the pages of a blog to extract the

DOM structure and parses the pages to extract the post data. Finally, we adapt the *DBUpdaterJob* by adding a blog-specific scoring mechanism (see section IV-E2).

V. Related Work

Certainly, the idea of crawling the *blogosphere* is not a novelty. However, the ultimate objectives and methods behind the different research projects regarding automated and methodical data collection and mining differ greatly as the following examples suggest:

While Glance et. al. employ a similar data collection method in the blogosphere as we do, their subset of data is limited to 100.000 weblogs and their aim is to develop an automated trend discovery method for the blogosphere in order to tap into the collective consciousness of the blogosphere [16]. Song et al. in turn try to identify opinion leaders in the blogosphere by employing a special algorithm that ranks blogs not only according how important they are to other blogs, but also according to how novel the information is they contribute [17]. Bansal and Koudas are employing a similar but more general approach than Song et al. by extracting useful and actionable insights with their *BlogScope-Crawler* about the 'public opinion' of all blogs programmed with the blogging system blogspot.com [18]. Extracting geographic location information from weblogs and indexing them to city units is an approach chosen by Lin and Halavais [19]. Bruns tries to map interconnections of individual blogs with his IssueCrawler research tool [20]. His approach comes closest to our own project's objective of leveraging (content-related) structures and dynamics of emerging networks within the blogosphere. His data set and project scope are however not as extended as ours, since he focuses on the Australian blogosphere that is concerned with debating news and politics.

Overall, it is striking that many respectable research projects regarding knowledge discovery in the blogosphere [21] [22] hardly make an attempt in explaining where the data - necessary for their ongoing research - comes from and how it is ultimately obtained. We perceive it as nearsighted to base research like the ones mentioned before on data of external services like Technorati, BlogPulse or Spinn3r [23]. We at least make the effort of setting up our own crawling framework to ensure and prove that the data employed in our research has the quantity, structure, format and quality required and necessary [24].

VI. Outlook

We described several concepts of developing an intelligent and flexible blog crawler. Currently, we are implementing the new concepts as described in section IV as an extended version of the *Apache Nutch* crawler based on an in-memory database. Thereby, we soon expect to get in short-term a representative data set at least for the german blogosphere. The next steps will be to adapt the existing analysis components to the new database and explore new analysis approaches through the in-memory database. In parallel, we are working on new visualization concepts corresponding to analyze the source of different discussions about certain topics. Hereby, it should be possible to track a hot topic from the first post on. As a result, the spreading through blogs, news and social networks of a topic get more understandable.

VII. Conclusion

To sum up, in this paper we describe the current status of the feed crawler project and three concepts to improve the original feed crawler prototype. Firstly, we discuss the move to a high parallel and scalable crawling using *MapReduce* and *Apache Nutch*. Secondly, we present the usage of an in-memory database to accelerate the job selection and the blog analysis. Finally, we introduce the parsing of html pages of a blog for post attributes using patterns that are automatically extracted from the blog's feed.

Generally, we try to investigate in what patterns, and to which extent blogs are interconnected. We also have great interest in analyzing the content of single weblogs. Due to that, we want to face the challenge of long-term mining the *blogosphere* on a global scale. The visualization of link patterns, a thorough social network analysis, and a quantitative as well as qualitative analysis of bidirectional-linked blogs will form the following project phase of our overall project [9], which will build upon the enhanced performance and data collection techniques described in this paper.

Even though the original implementation performed well along the milestones defined in the current crawler implementation, it soon became apparent that those enhancements discussed in our paper [25] were crucial for the overall performance of the crawling framework. We conclude that the new *blog crawler* will be universal and run on a performance level that satisfies the major requirements for long-term and large-scale data mining in the *blogosphere*. Due to the enormous amount of blogs currently around, as well as those thousands of blogs and posts that add up to this amount of data every day, a final performance analysis of the crawler will follow in a couple of month.

[9] http://www.blog-intelligence.com

REFERENCES

[1] S. L. B. S. Herring, S. and E. Wright, "Bridging the gap: A genre analysis of weblogs," in *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04)*, 2004.

[2] T. Smith. (2008, Sep. 2,) Power to the people: Social media tracker wave 3. [Online]. Available: http://www.goviral.com/articles/wave_3_20080403093750.pdf

[3] D. Sifry. (2006) State of the blogosphere. Sifry.com - Sifry's Alerts. [Online]. Available: http://www.sifry.com/alerts/archives/000443.html

[4] P. S. C. M. J. Bross, A. Acar, "Spurring design thinking through educational weblogging," in *Pro. 2009 IEEE International Conference on Social Computing, IEEE Press*, vol. 14, Aug. 29–31, 2009, pp. 903–908.

[5] *H. Kircher*, it - Information Technology (49) 1, Oldenbourg Wissenschaftsverlag Std., 2007.

[6] M. Ojala, "Blogging for knowledge sharing, management and dissemination," in *Business Information Review*, 2005, pp. 269–276.

[7] D. Whelan, "In a fog about blogs," in *American Demographics*, vol. 25, New York, NY, 2003, pp. 22–23.

[8] *J. Schmidt*, Weblogs Eine kommunikations-soziologische Studie, UVK Verlagsgesellschaft mbH Std., 2006.

[9] S. M. C. Leisegang, "Sieben frei verfgbare weblog-systeme liebes tagebuch..." in *Business Information Review*, 2008, p. 42.

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[11] R. Khare and D. Cutting, "Nutch: A flexible and scalable open-source web search engine," Tech. Rep., 2004.

[12] M. M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene." in *IPDPS*. IEEE, 2007, pp. 1–8. [Online]. Available: http://www.cecs.uci.edu/ papers/ipdps07/pdfs/SMTPS-201-paper-1.pdf

[13] H. Plattner and A. Zeier, *In-Memory Data Management: An Inflection Point for Enterprise Applications*, 1st ed. Springer, 4 2011. [Online]. Available: http://amazon.com/o/ASIN/3642193625/

[14] K. R. C. M. Justus Bro, Matthias Kohnen, *Identifying the top dogs of the blogosphere*. Springer LNSN, 2011, vol. Social Network Analysis and Mining.

[15] Z. Bar-Yossef, I. Keidar, and U. Schonfeld, "Do not crawl in the dust: different urls with similar text," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2007, pp. 111–120. [Online]. Available: http://dx.doi.org/10.1145/1242572.1242588

[16] N. S. Glance, M. Hurst, and T. Tomokiyo, "BlogPulse: Automated Trend Discovery for Weblogs," in *WWW 2004 Workshop on the Weblogging Ecosystem*. New York, NY USA: ACM, May 2004. [Online]. Available: http://www.blogpulse.com/papers/www2004glance.pdf

[17] X. Song, Y. Chi, K. Hino, and B. Tseng, "Identifying opinion leaders in the blogosphere," in *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. New York, NY, USA: ACM, 2007, pp. 971–974. [Online]. Available: http://dx.doi.org/10.1145/1321440.1321588

[18] N. Bansal and N. Koudas, "Searching the blogosphere." in *Proceedings of the 10th International Workshop on Web and. Databases, WebDB 2007*, Beijing, China, 2007.

[19] J. Lin and A. Halavis, "Mapping the Blogosphere in America." [Online]. Available: http://www.blogpulse.com/papers/www2004linhalavais.pdf

[20] A. Bruns, "Methodologies for mapping the political blogosphere: An exploration using the issuecrawler research tool," *First Monday*, vol. 12, no. 5, pp. 1109–1110, 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0190962207015885

[21] N. Agarwal, H. Liu, L. Tang, and P. S. Yu, "Identifying the influential bloggers in a community," in *WSDM '08: Proceedings of the international conference on Web search and web data mining*. New York, NY, USA: ACM, 2008, pp. 207–218. [Online]. Available: http://doi.acm.org/10.1145/1341531.1341559

[22] S. C. Herring, L. A. Scheidt, S. Bonus, and E. Wright, "Bridging the gap: A genre analysis of weblogs," in *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04)*. Los Alamitos: IEEE Press, 2004.

[23] M. Chau, J. Xu, J. Cao, P. Lam, and B. Shiu, "A blog mining framework," *It Professional*, vol. 11, no. 1, pp. 36–41, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4757239

[24] J. Bross, M. Quasthoff, P. Berger, P. Hennig, and C. Meinel, "Mapping the blogosphere with rss-feeds," in *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, ser. AINA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 453–460. [Online]. Available: http://dx.doi.org/10.1109/AINA.2010.95

[25] P. B. C. M. Justus Bross, Patrick Hennig, "Rss-crawler enhancement for blogosphere-mapping," in *International Journal of Advanced Computer Science and Applications,*, ser. IJACSA '10, 2010.