Collaborative Web Crawler over High-speed Research Network

Shisanu Tongchim and **Prapass Srichaivattana** and **Canasai Kruengkrai**
Virach Sornlertlamvanich and **Hitoshi Isahara**
Thai Computational Linguistics Laboratory
National Institute of Information and Communications Technology
112 Paholyothin Road, Klong 1, Klong Luang, Pathumthani 12120, Thailand
{shisanu,prapass,canasai,virach}@tcllab.org, isahara@nict.go.jp

Abstract

This paper proposes an idea for constructing a distributed web crawler by utilizing existing high-speed research networks. This is an initial effort of the Web Language Engineering (WLE) project which investigates techniques in processing the languages found in published web documents. In this paper, we focus on designing a geographically distributed web crawler. Multiple crawlers work collaboratively and use an existing research network for communication. Two methods for distributing the workload of web crawling are investigated. The first one divides the workload among these crawlers by selecting the nearest crawler to retrieve each domain, while the second one distributes the crawling task by using a hash function. As a prototype, we implement two crawlers, one is located in Thailand while another one is located in Japan. Some statistics about the prototype crawler are shown. In the end, we discuss about the use of collaborative crawling for the real-time monitoring of language usage in web documents.

1 Introduction

The web is a huge space of information that has been used in several studies in the computational linguistics field. To collect the data from such a space, we need a so-called web crawler to traverse the web and collect pages by following hyperlinks. In this report, we propose an idea for constructing a distributed web crawler by utilizing existing high-speed research networks. The motivation of this project comes from the increasing number of web servers and the dynamic nature of the WWW. To tackle with such a large web information space, we need a scalable solution. In this project, the crawler workload will be distributed across several servers. Each server that joins this system will contribute the data storage and bandwidth required for the web crawling. The communication between crawlers is done through high-speed research networks. With the expanding of research networks, it is expected that more participants will join this project in the near future.

2 Web Language Engineering Project

The Web Language Engineering (WLE) project is established to study about the language processing techniques for diverse, multilingual web documents. WLE needs a crawler to collect web documents which will be used in the language processing research. In the first stage of this project, the retrieved web documents have been used to test the language identification engine and multilingual information retrieval.

Our project differs from some projects like the Language Observatory Project (LOP) (Mikami et al., 2005). Although our project and LOP also employ a crawler to collect web documents for later studies, LOP intends to analyze the language usage in cyberspace rather than investigating the language processing techniques. LOP aims to provide the periodic statistics of languages, scripts, character encoding found in cyberspace in order to answer the questions like: How many languages are found in cyberspace? Which languages are missing? LOP also tries to find web pages of less spoken languages.

WLE focuses on the techniques to collect web documents and how to use them. With the expanding of Academic and Research Network, we propose the use of research network as the infrastructure of the collaborative crawler. The availability of research networks has established the cooperation between research institutes, universities and other academic organizations worldwide. By employing a crawler software, each participant to this project can be regarded as a data collector node in our crawler network. Research networks provide effective communica-

tion channels between spatial distributed web crawlers. The networks also help in sharing the retrieved web data among participants. Further applications can be developed on this framework.

3 Distributed Web Crawler

Some researchers have proposed distributed web crawling over Peer-to-Peer systems (Loo et al., 2004; Burkard, 2002; Singh et al., 2003). While those techniques offer some advantages in scalability and robustness, there are still some challenging issues for the web crawling and archival task. For example, the concept of P2P allows users to join and leave freely. With such dynamic behavior, this makes it more difficult for designing the system. Some existing data may be lost if we do not have an effective replication mechanism. Moreover, the concept of P2P is intended for a large number of participants, each with fairly available computational resources. It can be considered as a fine-grained system. If users would like to donate bandwidth and disk storage on their personal computers, they will decide to join the system. These machines are loosely connected since it is quite difficult to implement a fully connected topology. Each node maintains its own local routing table, and knows how to reach only some nodes in the system. To make an effective keyword search on the spatial distribution of collected web information is quite difficult. If we would like to construct a web archival system or keyword search, it is still questionable whether the P2P web crawler is really justified.

The concept of our distributed web crawler can be considered as a coarse-grained system. It will consist of a number of participants, each with more computing resources when compared with the P2P web crawler. Since the participated machines will be dedicated for this task, the dynamic joining and leaving will not be an issue. The system of each participant may be a cluster of workstations. Thus, the available resource will be more suitable for the web archival task. With the lower number of nodes keeping the web data, this make some applications like the keyword search through the entire system easier.

3.1 Job Distribution

Since multiple crawlers work together, a job distribution algorithm is necessary in order to avoid performing duplicate work. Some studies (Loo et al., 2004; Burkard, 2002) use some hash functions to map URLs or domains to nodes. The advantage of this scheme is that the calculation is straightforward and it is simple to implement. Moreover, this scheme does not need to keep the track of which node is responsible for a particular URL or domain. However, it does not consider geographical proximity of the web servers to crawlers. Thus, it cannot guarantee that a domain is crawled by the nearest node.

Another approach is to partition by the network proximity. Each domain is responsible by the nearest crawler. All crawlers will calculate a delay time for a given domain to find which crawler is the fastest to get web pages from this domain. In this paper, we conduct a comparison between both methods for job distribution. The descriptions of both algorithms are as follows:

1. *Hash algorithm*: When one crawler discovers a new URL, it will use a hash function to map the domain name of this URL to a node. This URL will be forwarded if another node is responsible to this URL. Since the crawling task is divided by domain names, every pages from the same web site or the same domain name will be fetched by the same crawler. Thus, the hash function will be used only when a new domain name is found.

2. *Network proximity algorithm*: A particular domain name is assigned to the nearest node from the network proximity information. All pages that belong to this domain name will be crawled by the assigned node. The implementation is straightforward. When one crawler discovers a new domain, it will send this domain to the others. Each crawler will measure the delay time to make a connection with the server of this domain. All nodes send the results to the first node. Then, the first node determines the crawler with the minimum delay time which will be responsible for downloading all web contents from this domain and informs this information to the others. If one crawler discovers a new URL that is already assigned to another node, there is a redirection mechanism in order to pass this URL on to another.

   This algorithm obviously utilizes more network bandwidth than that of the first algorithm. That is, three steps of communication have to be performed when a new domain

name is discovered. These steps have to be finished before the URL forwarding can be done. In contrast, the hash algorithm can determine the node that is responsible for a particular URL directly. Therefore, the hash algorithm utilizes network bandwidth only for the URL forwarding. Moreover, the network proximity algorithm also needs a table to store a list of already assigned nodes to domain names. Thus, it will use more memory than the hash algorithm.

3.2 Our Prototype

In the first stage of implementation, we test this system across two geographically distributed nodes. The first crawler is located in Thailand, while the second one is in Japan. We have collaboration with Prof. Yoshiki Mikami, the project leader of LOP. He allows us to use one machine in his laboratory as a test machine. Both machines are connected through the research network between Thailand and Japan. The communication between two sites is used mainly to distribute the crawling workload. The schematic diagram of our implementation is shown in Fig. 1. The crawler was implemented using multi-thread Java.

The main components of our crawler conform to those of a typical crawler. The primary components are as follows:

- *Fetcher*: The Fetcher is responsible for downloading web pages from the list of URLs. The Fetcher picks up the list of URLs from the queue. This queue contains all URLs that remain to be downloaded by the Fetcher. Every URL is checked whether it is already downloaded or not. If it is a new URL, it will be added to the queue. When a page is downloaded, its information will be marked to the Seen-URLs data structure in order to avoid a content duplication.

- *URL Extractor*: This component extracts URLs from a downloaded web page. The extracted URLs will be tested whether they are already downloaded (or already put to the queue) or not.

- *Duplicate Tester*: Before adding a new URL to the queue, it will be checked in order to avoid downloading an already downloaded document.

In addition to the standard components, our crawler also incorporates these modules.

- *Job distribution and the redirection mechanism*: Two algorithms for the job distribution are implemented. Both algorithms have the redirection mechanism to forward a particular URL to the node responsible.

- *Language identification module*: Every downloaded web page is sent to the language identification module called LIBS (Kruengkrai et al., 2005). The results are kept in the database for later showing on an on-line web interface. The language identification module is based on the concept of string kernels. It has been tested on two different kernel classifiers: the kernelized version of the centroid-based method and the support vector machines. Currently, our language identification module supports 61 different languages and 110 language-encoding pairs.

- *Real-time monitoring*: This module provides statistics of crawlers and statistics from the language identification module in terms of languages, encodings and the defined charsets.

4 Preliminary Results

An experiment is conducted on our prototype crawler in order to compare the job distribution algorithms and to answer the following questions:

- Is there any difference in the crawling speed between two algorithms?

- Is there any difference in the domain assignment between two algorithms? How different are they?

- Is there any difference in the network bandwidth consumption between two algorithms?

It is straightforward to answer the first and the third questions. To find an answer for the second question, we limited our crawlers to crawl only on *.jp* (Japan) and *.th* (Thailand) domains. Intuitively, it seems to be faster to access .jp domains from the Japan server and .th domains from the Thailand server, although this assumption may not be valid for every domain. The differences

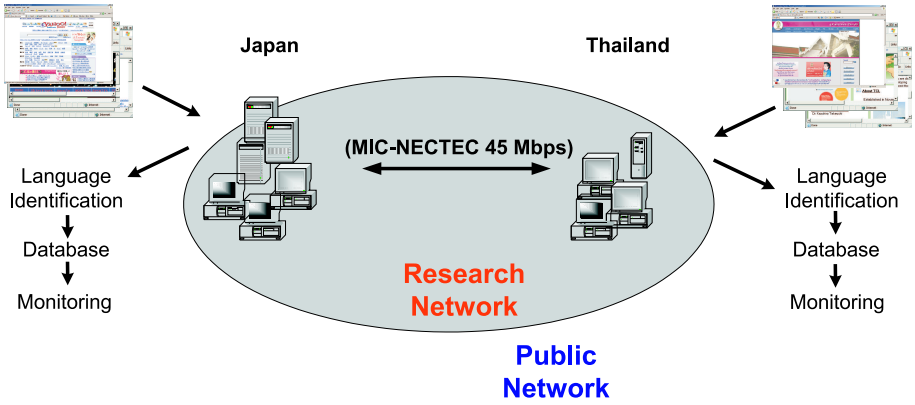

Figure 1: The first stage of implementation

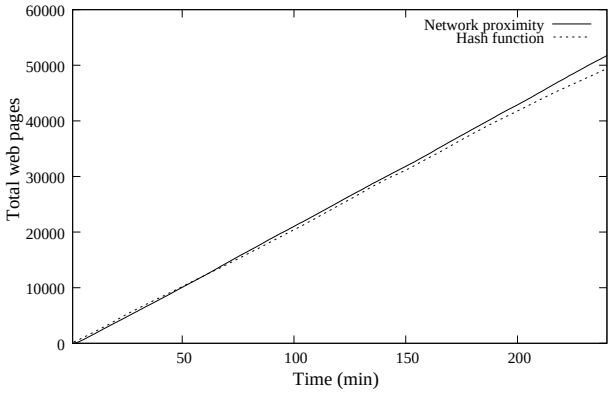

Figure 2: The number of web pages as function of time separated by the algorithms

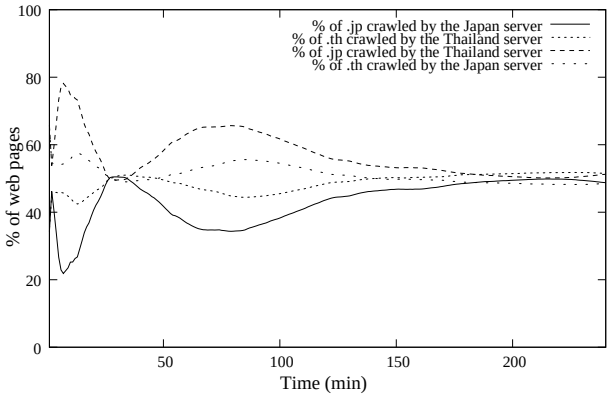

Figure 3: The percentage of web pages separated by the domains in each server using the hash algorithm

in the accessing time would show the differences between the domain assignments of two algorithms.

Note that the Japan server is forced to use a proxy server due to the university regulation. In order to avoid the cache effect, the crawler program asks the proxy server not to use its cache for every delay testing of the network proximity algorithm. In this regard the Japan server has more overhead than the Thailand server.

The results of each algorithm are averaged from three runs. The number of web pages plotted as function of time is shown in Fig. 2. This graph compares the crawling speed between two algorithms. From the graph, the average speeds of both algorithms are quite identical.

We analyze the percentage of web pages separated by the domains in each server. The results of crawlers using the hash algorithm and the network proximity algorithm are shown in Fig. 3 and Fig. 4 respectively. From Fig. 3, the links with .jp domains and .th domains are divided evenly between two crawlers after running for a period of time. From Fig. 4, the results show a different behavior of the network proximity algorithm. That is, the majority of .jp domains are assigned to the Japan server, while the majority of .th domains are assigned to the Thailand server.

To answer the last question, we calculate the percentage of communication overhead ($\mathcal{C}$) used in both algorithms. The percentage of communication overhead of the job distribution algorithm i is defined as follows:

$$\mathcal{C}(i) = \frac{o(i)}{o(i) + f(i)} \times 100 \qquad (1)$$

where $o(i)$ is the amount of bandwidth used for the job distribution algorithm i, $f(i)$ is the amount of bandwidth used for fetching web doc-

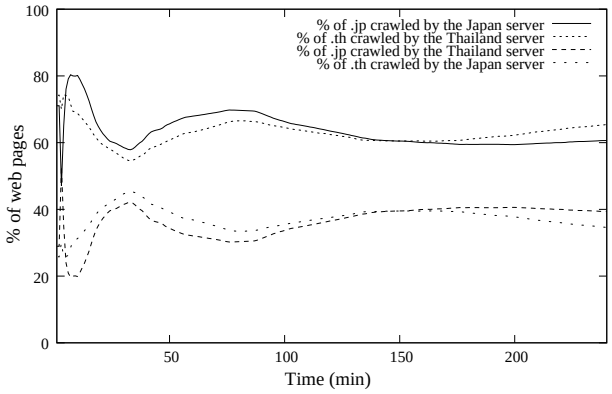

Figure 4: The percentage of web pages separated by the domains in each server using the network proximity algorithm

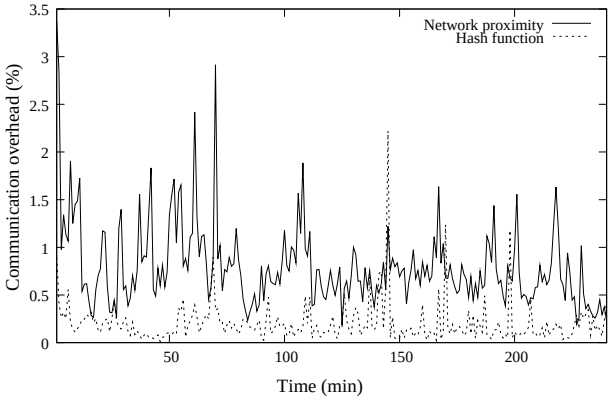

Figure 5: Percentage of communication overhead

uments.

The percentage of communication overhead reveals the amount of network bandwidth consumed by the job distribution algorithm in comparison with the network bandwidth used for retrieving web documents. Ideally, this value should be minimal.

The percentage of communication overhead between the crawlers is illustrated in Fig. 5. The graph shows that the network proximity algorithm utilizes more network bandwidth than that of the hash algorithm. Anyway, the communication overhead of both job distribution algorithms is relative small, with less than 3%.

The results reveal that two algorithms have differences in the domain assignment. Moreover, the network proximity algorithm also has more communication overhead. Nonetheless, both algorithms perform at the same speed for web crawling. From an algorithmic viewpoint, the network proximity algorithm has an advantage in selecting the faster crawler for a particular domain. However, the network proximity algorithm incurs more steps to perform the domain assignment and could reduce performance.

5 Real-time monitoring

As a part of WLE, an on-line web interface[1] is established to provide real-time monitoring of the collaborative crawling and statistics of retrieved web documents. Statistics of web documents in the aspects of language and encoding are obtained from the language identification module called LIBS (Kruengkrai et al., 2005). This approach can identify the language from the text directly (i.e. as a string of bytes), regardless of its coding system. LIBS identifies the language and encoding used in each web document and stores this information for later showing by the on-line web interface. Another information that is shown on the web interface is the defined charset. The defined charset can be obtained from the header of html documents. It is shown for comparison with the identified encoding by LIBS. An example screenshot of the web interface is presented in Fig. 6. The results shown in the web interface are separated by crawlers. Users can request statistics of a particular crawler or the entire system.

Recently, we develop a visualization of the language usage in web documents for Google^{TM}Earth. The results are produced as a Google^{TM}Earth KML file and can be opened by Google^{TM}Earth clients. The results are shown on a 3D globe. The web documents are separated by their countries of origins using Country code top-level domain (ccTLD), e.g. '.th' for Thailand. Users can request statistics of the language usage in a country. An example screenshot showing the results of Madagascar (.mg) is presented in Fig. 7. The visualization also provides some information about hyperlinks. Figure 8 shows the information of hyperlinks from Madagascar to web documents in other countries.

6 Conclusions

In this project, we propose a distributed web crawler that utilizes the existing research networks. Crawlers are geographically distributed and connected through high-speed research net-

[1] http://www.tcllab.org/weblang

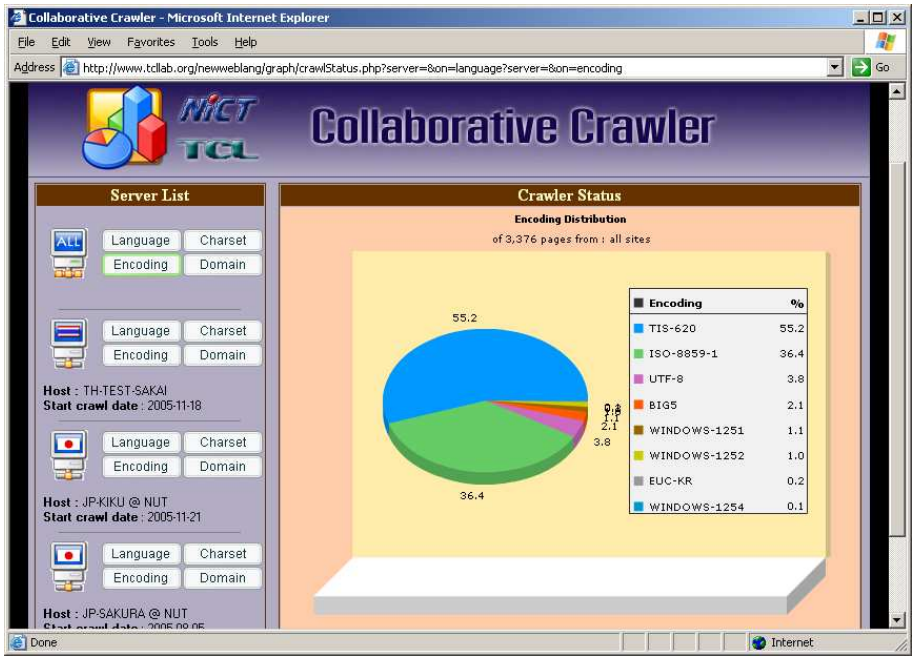

Figure 6: A screenshot of the real-time monitoring

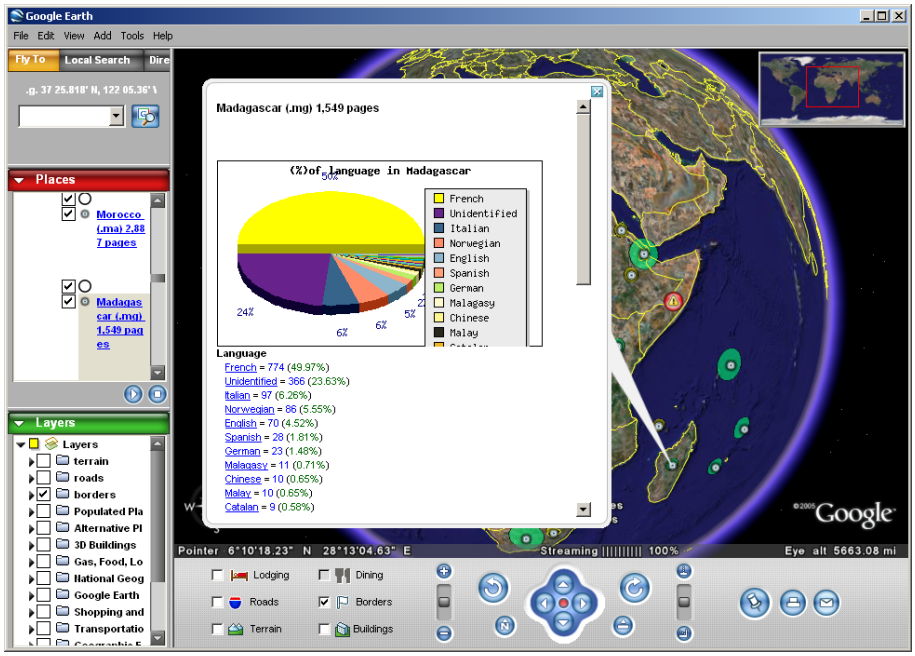

Figure 7: Statistics of web documents from Madagascar (.mg)

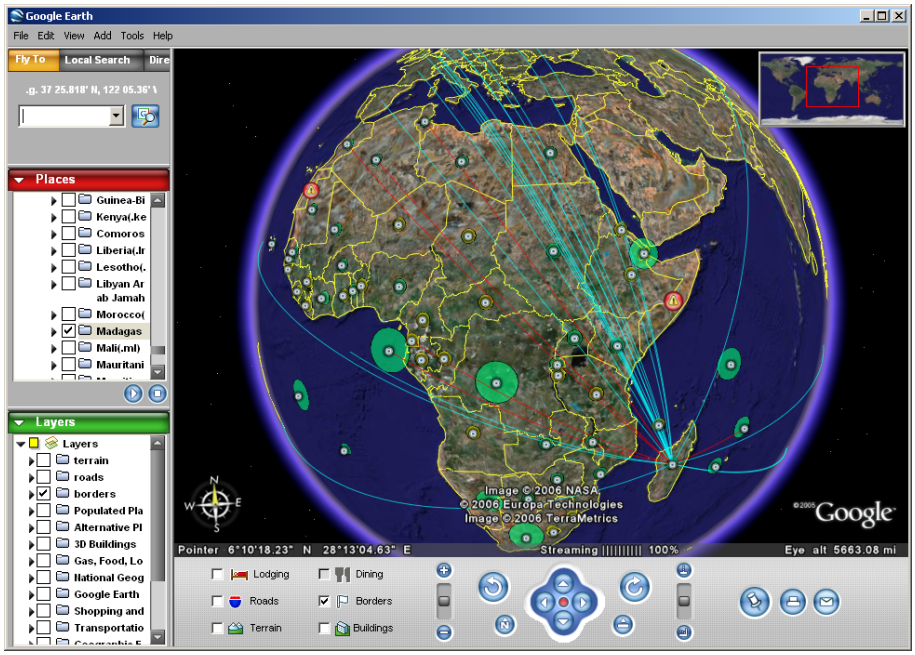

Figure 8: Hyperlinks from Madagascar (.mg)

works. Two algorithms for distributing the crawling workload are examined. The results show that both algorithms perform at the same speed, although they have different behavior in the job assignment. In addition, we discuss the use of collaborative crawler for monitoring the language usage in web documents. The results are shown in an on-line web interface and Google™Earth.

Acknowledgment

The authors would like to thank Mr. Kergrit Robkop and Mr. Suphanut Thanyaboon who implemented the real-time monitoring and the Google™Earth visualization.

References

T. Burkard. 2002. Herodotus: A peer-to-peer web archival system. Master's thesis, Massachusetts Institute of Technology.

C. Kruengkrai, P. Srichaivattana, V. Sornlertlamvanich, and H. Isahara. 2005. Language identification based on string kernels. In *Proceedings of the 5th International Symposium on Communications and Information Technologies*, volume 2, pages 896–899.

B.T. Loo, S. Krishnamurthy, and O. Cooper. 2004. Distributed web crawling over DHTs. Technical Report UCB-CS-04-1305, UC Berkeley.

Y. Mikami, P. Zavarsky, M.Z.A. Rozan, I. Suzuki, M. Takahashi, T. Maki, I.N. Ayob, P. Boldi, M. Santini, and S. Vigna. 2005. The language observatory project (LOP). In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 990–991, New York, NY, USA. ACM Press.

A. Singh, M. Srivatsa, L. Liu, and T. Miller. 2003. Apoidea: A decentralized peer-to-peer architecture for crawling the world wide web. In James P. Callan, Fabio Crestani, and Mark Sanderson, editors, *Distributed Multimedia Information Retrieval*, volume 2924 of *Lecture Notes in Computer Science*, pages 126–142. Springer.