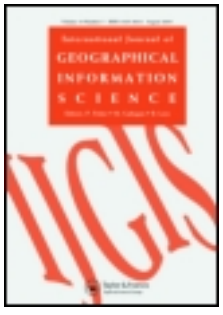

International Journal of Geographical Information Science

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tgis20

An active crawler for discovering geospatial Web services and their distribution pattern - A case study of OGC Web Map Service

Wenwen Li [a], Chaowei Yang [a] & Chongjun Yang [b]

[a] Joint Center for Intelligent Spatial Computing, College of Science, George Mason University, Fairfax, VA, USA

[b] Institute of Remote Sensing Applications, Chinese Academy of Sciences, Beijing, China

Version of record first published: 16 Jun 2010.

To cite this article: Wenwen Li , Chaowei Yang & Chongjun Yang (2010): An active crawler for discovering geospatial Web services and their distribution pattern - A case study of OGC Web Map Service, International Journal of Geographical Information Science, 24:8, 1127-1147

To link to this article: http://dx.doi.org/10.1080/13658810903514172

International Journal of Geographical Information Science
Vol. 24, No. 8, August 2010, 1127–1147

Taylor & Francis
Taylor & Francis Group

An active crawler for discovering geospatial Web services and their distribution pattern – A case study of OGC Web Map Service

Wenwen Li[a]*, Chaowei Yang[a] and Chongjun Yang[b]

[a]*Joint Center for Intelligent Spatial Computing, College of Science, George Mason University, Fairfax, VA, USA;* [b]*Institute of Remote Sensing Applications, Chinese Academy of Sciences, Beijing, China*

(*Received 1 February 2008; final version received 22 November 2009*)

The increased popularity of standards for geospatial interoperability has led to an increasing number of geospatial Web services (GWSs), such as Web Map Services (WMSs), becoming publicly available on the Internet. However, finding the services in a quick and precise fashion is still a challenge. Traditional methods collect the services through centralized registries, where services can be manually registered. But the metadata of the registered services cannot be updated timely. This paper addresses the above challenges by developing an effective crawler to discover and update the services in (1) proposing an accumulated term frequency (ATF)–based conditional probability model for prioritized crawling, (2) utilizing concurrent multi-threading technique, and (3) adopting an automatic mechanism to update the metadata of identified services. Experiments show that the proposed crawler achieves good performance in both crawling efficiency and results' coverage/liveliness. In addition, an interesting finding regarding the distribution pattern of WMSs is discussed. We expect this research to contribute to automatic GWS discovery over the large-scale and dynamic World Wide Web and the promotion of operational interoperable distributed geospatial services.

Keywords: geospatial Web service (GWS); crawler; Web Map Service (WMS); accumulated term frequency (ATF); conditional probability; clumped distribution

1. Introduction

The development of geospatial information acquisition methods helps to collect huge amounts of geospatial information. In 2006 alone, NASA's Earth Observing System Data and Information System (EOSDIS) produced over 3 terabytes (TB) of Earth system science data on a daily basis (NASA 2007). The geospatial information is widely utilized in different applications, such as navigation (Rae-Dupree 2006), transportation (Peytchev and Claramunt 2001), urban planning (Stevens *et al*. 2007), and emergency response (Rauschert *et al.* 2002). However, they are archived in various forms, and the geospatial applications, provided by different vendors, are highly heterogeneous in data representation, storage, and access (Paul and Ghosh 2006). The heterogeneity makes it difficult to share and exchange geospatial information.

To solve this heterogeneity problem and to facilitate better sharing of geospatial information, standards have been developed by a variety of organizations. In 1994 the Open Geospatial Consortium (OGC), the Federal Geographic Data Committee (FGDC), and the

*Corresponding author. Email: wli6@gmu.edu

ISSN 1365-8816 print/ISSN 1362-3087 online

DOI: 10.1080/13658810903514172
http://www.informaworld.com

International Organization for Standardization/Technical Committee 211 (ISO/TC211) were established to develop a series of specifications and standards, such as FGDC Metadata Content Standards (Nebert 2004), Web Map Service (WMS; de La Beaujardiere 2004), Web Feature Service (WFS; Vretanos 2005), and Web Coverage Service (WCS; Whiteside and Evans 2006). Most of the specifications leverage geospatial Web services (GWSs), also referred to as distributed geospatial information services (DGISs; Yang and Tao 2006), to facilitate the sharing of geospatial information. The GWS defines software component interfaces that provide access to geospatial information through Hypertext Transfer Protocol (HTTP)-based queries.

For example, the WMS defines the interface for accessing geospatial data uniformly from remote servers in a standard format, such as Portable Network Graphics (PNG) and Graphics Interchange Format (GIF), through HTTP. Three WMS operations are defined and used in the following sequence: (1) 'GetCapabilities' requests the service metadata; (2) 'GetMap' requests a static map according to given geospatial and other parameters; and (3) 'GetFeatureInfo' requests data of selected features. The three operations are issued to a WMS in the format of http://WMS_URL? Request=Operations&Service=WMS&Version=1.*.* through the HTTP POST or GET protocols. The procedure allows a WMS to integrate different geospatial information and services at the mapping level. Figure 1 shows a WMS link residing in a web page as a hyperlink.

Other GWSs are defined to address interoperability at different levels, such as how WFS supports interoperation among vector data encapsulated by Geographic Markup Language (GML) and how WCS supports the interoperation of geospatial data (such as GeoTIFF). The difference is that WMS provides an image map rather than raw data (De La Beaujardiere 2004). WFS and WCS provide raw data to the client, but they are not immediately visible as the data are generally geometric objects/features described by a set of vertices and slightly primitive (Vretanos 2005, Whiteside and Evans 2006). Therefore, the implementation logic of WFS and WCS at the client side would be more complex.

The increased popularity of these standards has led to an increasing number of GWSs becoming publicly available on the Internet. Recently, researchers have explored automatic assembly of different GWSs to build distributed geospatial information systems utilizing

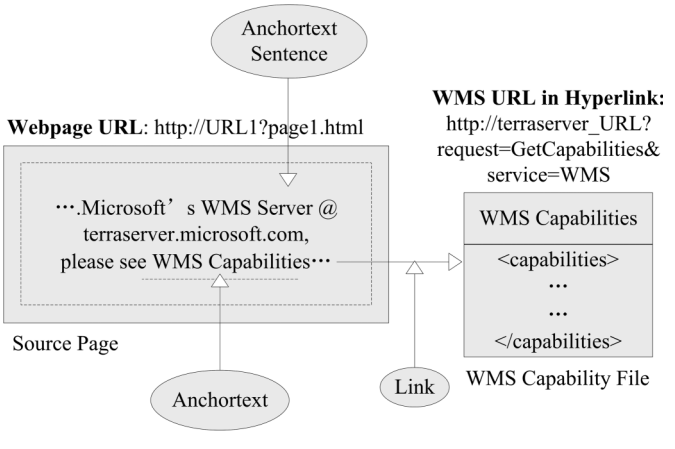

Figure 1. WMS's appearance on the Web: A web page may contain an anchor text with a Web link. The link may refer to a WMS URL for invoking a WMS capability file that describes basic service information for the WMS.

service chaining (Alameh 2003, Rajasekaran *et al.* 2004), Web service ontology modeling (Roman *et al.* 2005), and knowledge reasoning (Yue *et al.* 2007). Besides, international organizations are discussing potential specifications, such as OASIS ebXML Registry Profile for Web Ontology Language (Dogac *et al.* 2006) and OGC Catalogue Services-OWL Application Profile of CSW (Stock 2009), to enable semantic description of Web service feature types, properties, and science content in the registry to enhance keyword-based search. The prerequisite of all the above work is to have a significant number of live GWSs available (Keller *et al.* 2004). Therefore, discovering the services in the open and dynamic environment of the Internet becomes a critical task (Egenhofer 2002, Sample *et al.* 2006). Among several approaches for discovering GWSs, a centralized catalog with registered services is the most popular (Singh *et al.* 2003, Ma *et al.* 2006). The catalog approach does help users to discover GWSs; however, it is based on the premise that service providers have registered their services in the catalog and the services are registered with correct classifications and updated information. This assumption is frequently not met because many service providers do not register their services and many service metadata entries in catalogs are not updated in a timely fashion (Ran 2004, Al-Masri and Mahmoud 2007).

The second approach is to utilize popular search engines, e.g. Google, to discover the services. But the popular search engines aim to answer generic queries by treating all the keywords evenly without considering the characteristics of GWSs. Although the number of existing GWSs is considerable, they are still a very small portion compared to the information volume on the Web. And the ranking mechanism of Google is based on the number and weight of other links pointing to a certain web page, which is not measured by the quality of service (QoS). So if we only rely on Google to search, the exact service will be flooded and hidden in a long list of search results. Researchers have done experiments on this and found that the Googled WMS, WFS, and other accessible GWSs significantly underrepresent the available OGC services (Reichardt 2005). Moreover, the GWSs always exist on web pages that are geospatial related, whereas Google's method is to crawl the entire Web, which is not necessary here.

Observing the shortcomings of the two previous approaches, the development of an efficient domain-specific crawling algorithm and the implementation of such a crawler become a compelling solution (Wöber 2006). A domain-specific crawler can improve the search ability in both technical and economical perspectives. Technically, it is becoming increasingly difficult if not impossible for the general crawler to index the entire contents of the Web (Fesenmaier *et al.* 2006). It is also an economic hurdle to build such a large-scale index database for a crawler. In addition, a domain-specific crawler can improve the quality of searches in terms of (1) allowing searching of pages that are currently not searchable from the general search engines, (2) providing a more up-to-date search, (3) providing improved accuracy and extra features not possible with general search engines (Steele 2001). In the field of automatic WMS discovery, there is a so called 'WMS-Crawler', which crawls on the Web to find a hyperlink indicating a WMS and tries to parse it with a WMS capabilities analyzer. This approach is promising; unfortunately, there are very few WMS-crawlers implemented and the performance of existing crawlers (RR[1]; Sample *et al*. 2006, Schutzberg 2006) is not satisfying. This paper focuses on the research of the GWS crawling algorithm to (1) address the shortcomings of both current catalogs and general search engines; (2) propose an accumulated term frequency (ATF)-based conditional probability model to allow fast and automatic discovery and update of GWS on the Web; and (3) implement a high-performance search engine that can be easily integrated into catalogs and a QoS environment to support future automatic and smart service chaining. The discovery of WMSs is taken as an example.

2. Related work

The most notable research conducted in developing crawlers to discover WMSs on the Internet are Refractions Research (RR)'s WMS Crawler, GIDB WMS-Crawler (GIDB; Sample *et al.* 2006), and a WMS-crawler by Skylab Mobilesystems Ltd (Skylab; Schutzberg 2006).

RR utilizes the Google application programming interfaces (APIs) to find GWSs by searching WMS operation strings. It also supports the discovery of WFS and WFS-G (Gazetteer Service). However, the method of using Google APIs has inherited the limitation of Google's internal crawler and indexer: some recently available WMSs that have not been crawled by Google cannot be found by the Google API search.

The GIDB WMS-Crawler also leverages, but does not completely depend on, the Google APIs. GIDB uses GIS Web sites as seed URLs to reduce the number of web pages to be processed. It then extracts the URLs and sends GetCapabilities request to determine if a found URL is a real WMS by validating the response. Some technical approaches, such as filtering out 'none' XML responses, are followed to improve the performance. GIDB has advantages over the RR by using these techniques, but it does not designate priority for seed URLs during crawling. Therefore, even the discovery starts from several GIS Web sites. The out-links can still go far beyond the geospatial pages, where WMSs are embedded.

Skylab Mobilesystems Ltd. (Skylab) has developed a WMS-crawler, which is a commercial product, without sharing the technical design or performance evaluations. Therefore, it is hard to tell whether Skylab designed its own strategy or algorithm to crawl or utilized Google APIs. The known information is that the number of WMSs found through the crawler is 904 (both live and dead)[2], which is apparently below the real number.

This paper is trying to address the issues related to current WMS-crawlers to improve their performance in the following aspects by developing a new WMS crawler: (1) Efficiency: We will try to improve the discovery speed using multi-threading and other process management techniques. Since it is not realistic and necessary to crawl the entire Web, the goal is set to identify as many WMSs as possible within a specific time period. (2) Effectiveness: We will propose a special approach to focus on geospatially related pages to narrow the crawling scope by deciding which set of URLs has higher possibilities of containing or linking to WMSs to minimize the required Web crawling. (3) Updates: Since the availability of WMS is dynamic because of maintenance, a mechanism will be designed to keep the service metadata up to date.

The rest of the paper is organized as follows: Section 3 introduces the architecture design of our crawler, Section 4 presents the ATF-based conditional probability model and other techniques we used to improve the crawler, Section 5 introduces the crawler interface, workflow, and technical implementation, Section 6 compares the performance of our crawler to other existing crawlers, analyzes the results, and discusses the importance of findings on distribution pattern of GWSs, and Section 7 concludes the paper, discusses the universal problem in automated service discovery and possible solutions, and points out future research directions.

3. Crawler architecture

The crawler developed for WMS (Figure 2) includes crawling entry, buffer, source page analyzer, filter, repository, request/response (R/R handler), XML resolver, and a database to archive the WMS and relevant metadata discovery.

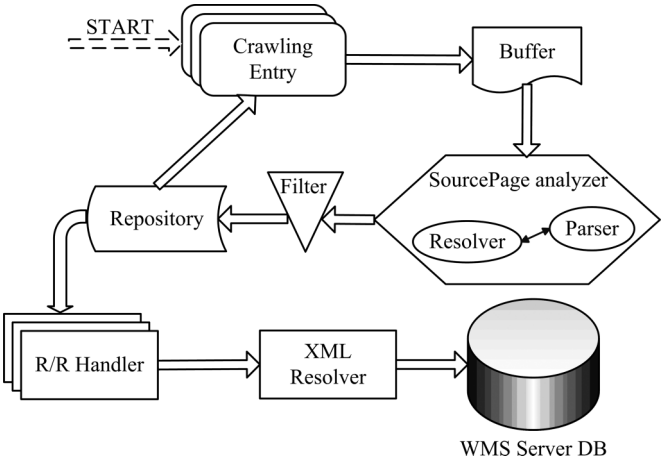

Figure 2. Crawler architecture (Li, 2007).

Crawling entry is where the crawler starts to work. One or more seeding Web URLs are either identified by users or fetched automatically from the repository. The concurrent multi-threading technique detailed in Section 4.3 is used to crawl the Web for improving efficiency.

Buffer selectively caches Web source code linked by URLs. Different from other generic crawlers which cache all the crawled Web documents for revisit, our buffer only caches those having the possibility of containing a WMS URL due to WMS's characteristics: web pages not containing WMS URLs when being crawled will have a low possibility of containing a WMS URL later on. Therefore, they are discarded. The hyperlinks cached in the buffer would be exported and stored to hard disk periodically. Once an unexpected error occurs, the crawling process can be recovered by loading the recorded information back to the memory. This buffering strategy helps to reduce memory waste, lower the maintenance cost, as well as tolerate fault during crawling.

Source page analyzer is used to analyze the Web source code that has been cached in the buffer. It extracts all out-links and transforms the relative URLs of the links into absolute URLs. After completing the process, the analyzer removes the source code based on the strategy adopted for the buffer module. Then the URL goes to the filter, which utilizes priorities based on the probability of linking to a WMS before putting URLs into the repository. The filter also filters out the URLs that already exist in the URL repository.

The *repository* maintains the URLs crawled using a priority queue, which are divided into several segments based on importance. *R/R (request/response) handler* gets the URL at the head of the repository and sends the WMS GetCapabilities requests to the server specified by the URL. The responses are parsed by the *XML resolver* to validate the URL's status using WMS tags, such as <WMS_Capabilities/>, <Service>, and <layer>. Once a WMS is found, metadata information such as layer numbers, service bounding box, spatial coverage, supported mapping projections, as well as other QoS-related information, including availability (whether a Web service is present or ready for immediate use) and latency (the round-trip time between sending a request and receiving the response), will be extracted and stored in the service catalog. The automatic metadata update will update the catalog periodically by a complementary model detailed in Section 4.4.

Section 4 details the improved techniques adopted for the crawler compared to the crawlers reviewed in Section 2.

4. Techniques for improving WMS crawler

4.1. Prioritized crawling: an ATF-based conditional probability model

It is very inefficient for a WMS crawler to visit all of the web pages on the Internet only to find a tiny part of resources that are needed. Therefore, quicker access to a web page with a WMS URL will improve the performance. This can be achieved by assigning crawling priorities to web pages and URLs. Given a URL u, the probability of u referring to a WMS can be determined as follows.

Definition 1: Given a URL u, we define the probability of u as $P(u) = (\mathrm{UP}, \mathrm{WP})(u)$, where UP($u$) denotes to URL priority and WP(u) denotes to priority of the web page from which u is extracted; both UP(u) and WP(u) have the same range $\{0, 1, 2, 3\}$.

The URL priority UP(u) is determined according to the characteristic of a WMS. The WMS is exposed in the format of a URL, which can handle a client's request through KVP (keyword value pair) encoding; thus the URL must be linked to an active web page instead of a static web page. Therefore, URLs linking to static web pages would not be considered as WMSs. But considering that the static web pages that the URLs link to might contain information related to a WMS, we still reserve them but assign them the lowest priority, UP(u) = 3. Some well-known image/document/video/audio formats can be excluded.

Higher priority is given to the active pages. An ATF analysis is performed and the initial priority levels (UP(u) = 0, 1 or 2) are assigned based on the elementary statistical results. The statistical priorities change dynamically when more WMSs are identified. To obtain this, the system maintains a list which contains terms (atomic substrings extracted from URL string) in descending order of their frequencies. Initially, based on the first N (we chose N = 50) WMSs found, we extracted the terms from their URL strings and calculated the TF values. Then the list was divided into three equal parts: the top one has the highest priority (Pr(t) = 0, t is any term in the top part), the middle one has the second highest priority (Pr(t) = 1), and the bottom one has the priority Pr(t) = 2. Each time a URL u is crawled, all the atomic substrings of the URL except protocol/host name are extracted, represented by $T_u = \{t_1, t_2 \ldots t_n\}$. The priority WP($u$) is determined by the highest priority of term t_i in T_u, namely $\mathrm{UP}(u) = \max\{\Pr(t_1), \Pr(t_2) \ldots \Pr(t_n)\}$. If there is no occurrence of t_i in the list, the priority is assigned as $\Pr(t_i) = 2$. If this URL is later justified as a WMS URL, all terms will be extracted and both the term occurrences and frequencies will be updated. For example, suppose list L contains five terms in current stage, which is L = {'request = GetCapabilities' (t_1), 20; 'service = WMS' (t_2), 16; 'version = 1.1.0' (t_3), 10; 'cgi-bin' (t_4), 3; 'servlet' (t_5), 3} in descending order of TF value. According to the section partition, the top two of them will be given the highest priority (0), namely $\Pr(t_1) = \Pr(t_2) = 0$, the following two have priority $\Pr(t_3) = \Pr(t_4) = 1$, and the last one has lowest priority among the current terms, $\Pr(t_5) = 2$. When a new URL u_x (http://hazards.fema.gov/wmsconnector/wmsconnector/Servlet/NFHL?request=GetCapabilities&service=WMS) is retrieved, the atomic substrings are extracted as T_{u_x} = {'wmsconnector', 'wmsconnector', 'Servlet', 'request = GetCapabilities', 'service = WMS', 'NFHL'}. According to the definition of *UP*, we obtain that $UP(u_x) = \max\{\Pr(t_i), t_i \in L\} = \min\{2, 2, 2, 1, 0\} = 0$. As u_x is testified as a WMS URL later on, all the terms in T_{u_x} will be combined with those in L. The new L with updated TF is L = {'request = GetCapabilities' (t_1), 21; 'service = WMS' (t_2), 17; 'version = 1.1.1' (t_3), 10; 'servlet' (t_4), 4; 'cgi-bin' (t_5), 3; 'wmsconnector' (t_6), 2; 'NFHL'

(t_7), 1}. Partitioning L into three sections, the priority for each term is obtained as$Pr(t_1) = Pr(t_2) = Pr(t_3) = 0$, $Pr(t_4) = Pr(t_5) = 1$, and $Pr(t_6) = Pr(t_7) = 2$, which would be used for future *UP* judgment. By a continuously learning process, the size of L increases gradually to generate an enriched controlled vocabulary, which contains fundamental semantic information to guide future crawling jobs.

Based on Definition 1, the priority for URLs can be formalized as

$$P(u_1|UP(u_1) = i) > P(u_2|UP(u_2) = j), \quad \text{where} \quad i,j \in \{0,1,2,3\} \quad \text{and} \quad i < j \qquad (1)$$

which means that if the URL priority of $u_1$is higher (smaller in value) than that of u_2, the probability that $u_1$is a WMS URL is larger than the probability that u_2 is a WMS URL.

By investigating web pages that may contain a WMS URL, the probability of web pages containing keywords [Web Map Service] (we define WP(u) = 0) is higher than that of those containing [WMS, Service] (defining WP(u) = 1), which is higher than that of those containing [WMS, Server] (defining WP(u) = 2), which is higher than any others that do not contain these keywords (defining WP(u) = 3). To verify this, we collected the top 400 web pages returned from the Google search engine with the above query keywords, and tests show that 16% of the web pages that contain [Web Map Service] contain a WMS URL in the source code, and the ratio is 13.25% and 10% for those containing [WMS service] and [WMS server], respectively. Those without the keywords have a ratio of having a WMS URL near zero. So the priority of a web page that may contain a WMS URL can be formalized as

$$P(u_1|\mathrm{WP}(u_1) = i, \mathrm{UP}(u_1) = k) > P(u_2|\mathrm{WP}(u_2) = j, \mathrm{UP}(u_2) = k), \quad \text{where} \quad i,j,k \in \{0,1,2,3\} \quad \text{and} \quad i < j \qquad (2)$$

This means that when the UP of two URLs is equal, the less WP a URL is, the higher probability it has to refer to a WMS.

4.2. *Priority queue*

A new URL found from web page out-links will be inserted into a priority queue (Figure 3) based on UP and WP defined by Equations (1) and (2). The URLs in the front of the queue have higher priorities and the queue is dynamic when inserting new URLs: UP decides which section the newly crawled URL belongs to, and WP decides the position of a URL in a section. For URLs with the same priority, the latest crawled URL will be inserted behind the

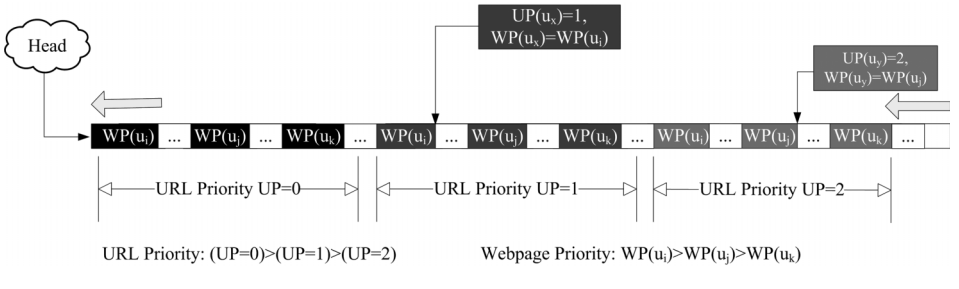

Figure 3. Data structure of the priority queue.

ones crawled previously. By this strategy, the crawling path is determined and recorded dynamically into the priority queue.

4.3. Multi-thread

Crawling web pages to find URLs and analyzing web pages to find WMSs are the most time-consuming components of the crawler. The processes are independent from each other except when they are trying to access the same section of the queue. Therefore, two groups of multi-threads can be utilized to speed up the crawling and analysis respectively: (1) Group 1 crawls the WWW, extracts URLs, and inserts the extracted URLs into a priority queue; and (2) Group 2 picks URLs from the priority queue and sends GetCapabilities requests to determine whether the URLs refer to live WMSs and updates the catalog. The priority queue is set as a critical section for the two thread groups. The concurrent threading could speed up the crawling but there is an optimal number of threads for performance tested in Section 6.1.

4.4. Automatic update

The automatic update module (Figure 4) is adopted to keep the WMS metadata up to date in the catalog and resolve the metadata for service and layers. It works in two modes: (1) Message driven: when a WMS is retrieved by the crawler, the automatic update module will be invoked to resolve metadata; (2) Revisit control: periodically, the module will pull the records out from the catalog and communicate with WMS servers to get the updated metadata. If the server is unavailable, the module will record it as an error and update the 'reason of unavailability' column in the catalog. The module could be added to *scheduled tasks*, so the operating system can invoke the module periodically to revisit and update WMSs' status. The revisit period is chosen by users and can be of any length, depending on the liveliness requirement of various application systems. The WMSs typically do not change as frequently as news Web sites do. Therefore, the update can be conducted as infrequently as once per day. Meanwhile, separating this module from the crawler architecture helps to isolate this time-consuming process from the crawling tasks, so the automatic status update is able to be conducted in parallel with crawling process. In this way, crawling efficiency is improved and the design complexity is reduced.

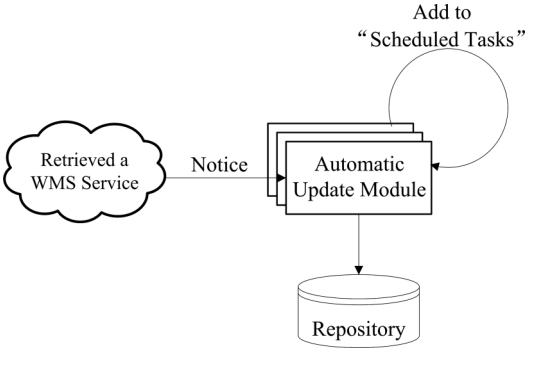

Figure 4. The WMS metadata automatic update module.

4.5. Algorithm description

Figure 5 depicts the crawler's main procedure that manages two thread groups: CrawlWebPage and CheckWMSSite. Data structure URL_queue is the priority queue. The crawler's URL priority determination is implemented by function Priority() in Figure 6. Resolving the XML file returned from a WMS is implemented in function Resolve() in Figure 7. The two thread groups maintain two cursors to the priority queue: one is for crawling the Web, and the other is for checking WMS URLs. Any group that reaches the end of the URL_queue will be suspended until a new URL is inserted.

Algorithm 1: Main Procedure

```
BEGIN
    EnterQueue (URL_queue, initial_URL, (0, 0))
    FOR i=0 TO (Max_Page_Threads-1) STEP 1 DO
        BeginThread (CrawlWebPage, thread_priority)
    END FOR
    FOR i=0 TO (Max_WMS_Threads -1) STEP 1 DO
        BeginThread (CheckWMSSite, thread_priority)
    END FOR
END
```

Figure 5. Pseudocode of main procedure.

Algorithm 2: ThreadGroup1

```
FUNCTION CrawlWebPage
BEGIN
    WHILE NOT all_been_visited (URL_queue)
    BEGIN
        URL = GetHeadCursor_Web (URL_queue)
        Webpage = crawl_page (URL);
        MarkAsCrawledPage (URL_queue, URL)
        url_list = extract_urls (Webpage)
        FOR EACH u IN url_list
        IF u NOT IN URL_queue THEN
            (p0_u, p1_u) =Priority (u)
            EnterQueue (URL_queue, initial_URL, (p0_u, p1_u)))
        END IF
        END FOR
    END WHILE
END FUNCATION
```

Figure 6. Pseudocode for crawler Thread group 1.

Algorithm 3: ThreadGroup2

```
FUNCTION CheckWMSSite
BEGIN
    WHILE NOT all_been_visited_for_WMS (URL_queue)
    BEGIN
        URL = GetHeadCursor_WMS (URL_queue)
        ReturnedFile=SendGetCapabilitiesRequest(URL);
        IF ReturnedFile CONTAINS WMS_Metadata_Tags THEN
            PutInDatabase (Resolve (ReturnedFile))
        END IF
    END WHILE
END FUNCATION
```

Figure 7. Pseudocode for crawler Thread group 2.

5. Prototype implementation

We implement proposed techniques into a prototype based on the Microsoft .NET framework. The main goal of the prototype is to enable an end user to customize the process of WMS crawling and to monitor system resource usage during crawling. The prototype is open source and available through an SVN server. Major technical details are as follows:

- Crawler is coded using Visual C++ and the graphic user interface (GUI) of the crawler is created using Microsoft Foundation Class (MFC).
- Automatic update model is coded using Visual Basic Script and is made executable using Windows Task Scheduler.
- Backend database adopts Microsoft SQL Server 2005 SP1 and its connection with the crawler is through Microsoft's Open Database Connectivity (ODBC) driver.

Figure 8 shows the GUI designed for a general user to utilize the prototype of our crawler. The top to bottom of the left side of the GUI is designated as the start point of the crawler: crawler status reports, including start time and duration of the crawling task; the number of crawling threads of the two groups; search results, including web pages that have been crawled and the WMS address if any WMS is found. The right side of the GUI shows statistical information including average processing speed and hardware usage for monitoring the crawler's status.

The sequence diagram of the crawler can be illustrated in Figure 9.

(1) Designate the start URL(s) and send requests to the crawler.
(2) Crawler sends HTTP GET Request to remote web server.

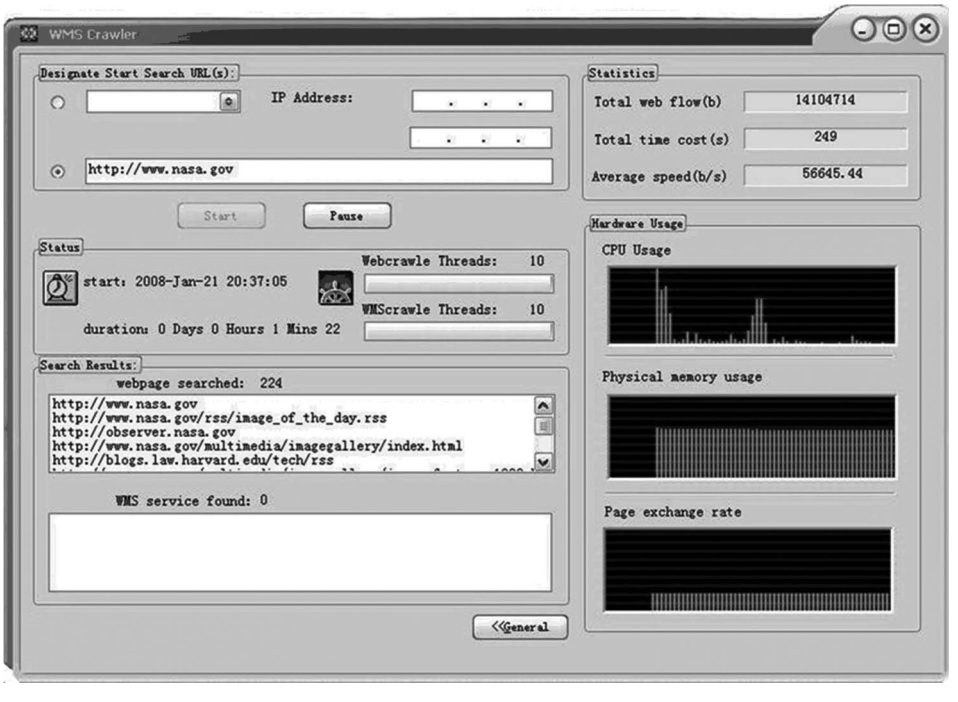

Figure 8. GUI of the crawler.

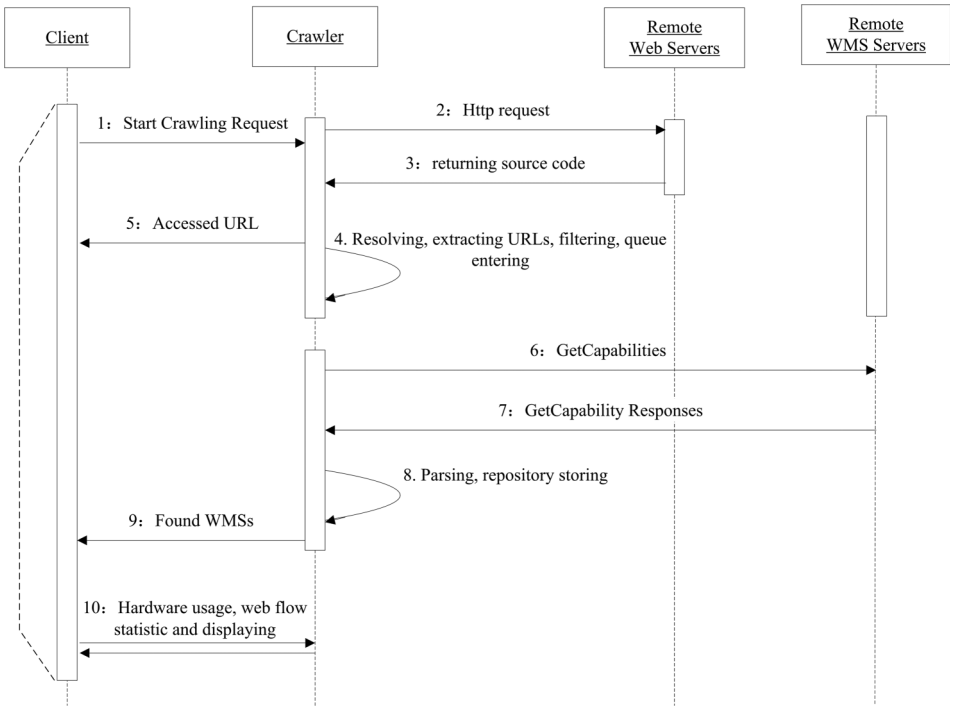

Figure 9. User sequence diagram.

(3) Return source code to the crawler.
(4) Crawler resolves the source code, extracts all the URL addresses that exist on the web page as hyperlinks or common text, then puts them into priority queue according to their priority.
(5) Crawler returns the accessed URL address to the client side for display.
(6) If crawler finds a similar WMS service, it sends GetCapabilities request.
(7) Capability file returns to crawler.
(8) Determines whether the URL is linked to a WMS server.
(9) Return the WMS URLs to client for display.
(10) Client and crawler communicate to get the status information while crawling.

For these functions, (2)–(5) are accomplished by Thread group 1 in modules 'source page analyzer' and 'filter'. Steps (6)–(9) are accomplished by Thread group 2 in modules 'R/R handler' and 'XML resolver'.

6. Crawler evaluation

This section compares the performance of our crawler to that of other popular crawlers, and evaluates how well our crawler performs with and without the proposed algorithm. The tests were conducted on a computer with a Pentium® 4 3.4 GHz CPU, 4 GB of RAM, and a 100 Mbps Internet connection. The test includes three parts: (1) efficiency (2) coverage and timeliness, and (3) precision.

6.1. Efficiency improvement by concurrent threads

Since the operating system (OS) only allocates limited CPU cycles and memory to execute a thread, it is very hard to speed up a single thread within a single program. Multi-threading, which increases the utilization of a single CPU core, leverages thread-level parallelism (Yang *et al.* 2005). Especially when a thread gets a lot of cache misses, the other threads can take advantage of unused computing resources to continue the crawling task. This leads to faster overall execution and higher system throughput. However, multiple threads also interfere with each other due to the sharing of hardware resources and thus generate critical inter-thread communication overhead. The optimal number of threads is a balance between the tasks and the computing resources including CPU, memory, hard disk, and network bandwidth. In Section 4.3, we discussed the threading model, in which one group of threads performs crawling tasks, and another group of threads performs determination tasks.

Figure 10 demonstrates that (1) when there is only one crawling thread and one determination thread, the crawler's throughput is very low; and (2) when we increase the number of crawling and determination threads to 10 each, the crawler's throughput increases dramatically. Through successive tests, we determined that having 72 crawling threads and 72 determination threads optimized the throughput (of course, this result only applies to the tested computing environment). Equal numbers of threads were chosen for crawling and determination because our initial experiments showed that this approach optimized the throughput.

It is also important that a crawler behaves politely to remote servers while maintaining a high crawling speed. If a crawler is performing many requests per second, the performance of a server could be negatively impacted. When calculating the crawling speed from the optimized thread groups (72, 72) obtained from the above experiment, our crawler averages 0.5 s per page, which is consistent with the polite access interval (1-s-per-page result) proposed by Dill *et al.* (2002). In addition, the priority determination mechanism introduced in our crawling algorithm helps the crawler to gather high-priority pages first. Unlike general search engines that tend to crawl a web page hosted by a server regardless of its format and

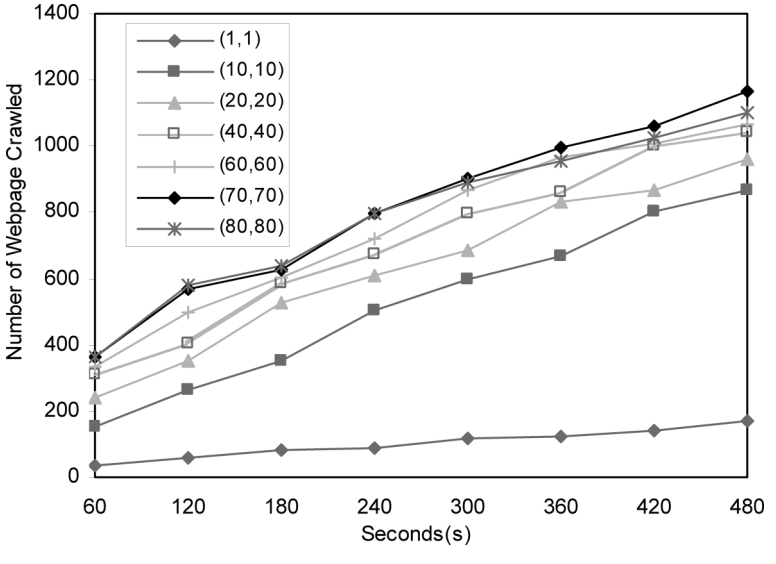

Figure 10. Crawling speed with different numbers of concurrent threads.

content, the proposed crawler restricts its access to WMS-relevant web pages by ATF matching. Both the crawling policy and observation from real-world experiments verify that our crawler acts politely to remote web servers.

6.2. *Coverage and timeliness compared to other WMS crawlers*

Coverage and timeliness of our crawler are compared to those of RR, Skylab, and GIDB[3]. The other crawlers' results (a list of WMSs) were obtained from their official site, the duplicates and dead links were removed, and non-WMS results (such as WFS, WCS, or WPS) were filtered out because we are only interested in comparing WMSs. The 'liveliness' of services and layers was determined by the downloading and parsing capabilities of WMSs. (We use the term 'liveliness' to denote whether the WMS referenced by a URL is actually alive and accessible.)

Figure 11 shows, for each crawler, the claimed number of WMSs found, the actual number of live WMSs found, the number of unique WMS hosts, and the total number of live layers. As of December 2008, our crawler had the best performance, with 1126 WMSs found. In decreasing order of performance, RR found 805 WMSs, Skylab found 701 WMSs, and GIDB found 611 WMSs. As RR, Skylab, and GIDB heavily rely on Google's API and only match the specific format of 'request = GetCapabilities' without considering other characteristic of a WMS, these crawlers fail to recognize many valid results even though they were found by Google. Our crawler is different in that it takes into account the Web locations of WMSs and the features indicated by their URLs in order to create a conditional probability–based model to locate WMS quickly and accurately.

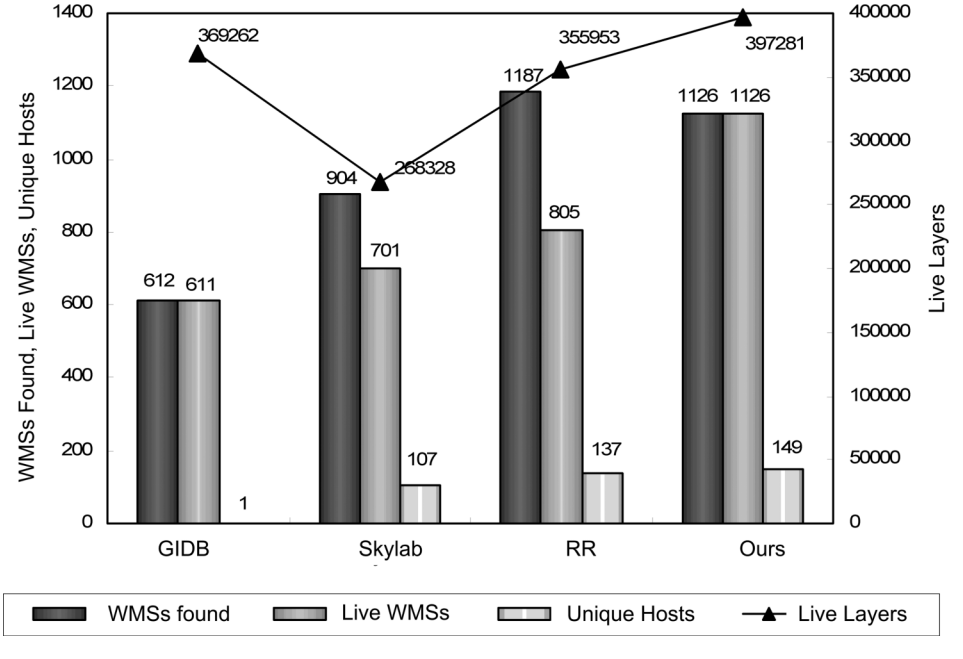

Figure 11. Comparison of different crawlers' (GIDB, Skylab, RR, and our crawler) coverage of WMSs.

In terms of timeliness, all of the other crawlers that we tested had a significant number of dead links in their results. When employing these results into practical applications, the system's performance would drop dramatically due to dead WMS reports, which is a great drawback of existing crawlers. In contrast, the liveliness of WMSs found by our crawler is 100%. Our crawler's automatic updating algorithm, described in Section 4.4, enables the high liveliness rate.

Interestingly, although our crawler discovered more live WMSs than the RR crawler did, the number of unique hosts where the services were located was almost the same for both crawlers. An explanation is that Google (RR relies on Google API) has the advantage of a very large-scale cache which can be easily searched, even though it is not designed for domain-specific crawling. However, Google limits its crawl depth by its page rank (PR) value (Sergey and Page 1999), so some web pages that have higher WP or UP but lower PR will not be crawled. In contrast, our crawler is better at analyzing and fully extracting WMSs from web pages in a single host. The reason for this is that our algorithm considers the characteristic of WMSs and employs a text-analyzing technique combined with a pure hyperlink analysis.

6.3. *Quickness in locating WMSs and findings regarding WMS distribution*

The priority queue based on a conditional probability model is adopted to make our crawler find WMSs more quickly. This experiment evaluates the proposed algorithm in comparison to a pure FIFO algorithm. The parameter 'quickness' (formula [3]) is used to measure how fast a crawler is able to identify a WMS.

$$\text{Quickness} = \frac{\text{Number of WMSs Found}}{\text{Total Webpage crawled}} \tag{3}$$

Notice that the quality of seed URLs will greatly affect the experimental results. Bad seeds may lead to a very time-consuming crawling without any WMSs found. To conduct a meaningful crawl, the crawler could start from a popular geospatial Web site (retrieved from general search engines by the keywords provided in Section 4.1), or it could start from a web server that is hosting a WMS or that has hosted WMSs before (as given by results of existing crawlers). Choosing one of these servers as a starting point is useful because it is more likely for such servers to host new WMSs or links to other Web sites that host a WMS. In this experiment, the seeds that were chosen are the following: (1) National Aeronautics and Space Administration (NASA): http://www.nasa.gov, (2) National Oceanic and Atmospheric Administration (NOAA): http://www.noaa.gov, and (3) Open Geospatial Consortium (OGC): http://www.opengeospatial.org. Among these seeds, both the NASA and NOAA seed URLs are located on the crawling path of the OGC seed, which means that all the WMSs that are found by starting from NASA and NOAA's web pages can be retrieved by starting from the OGC Web site. Here, extracting them out as independent seeds makes the discovery process more effective by avoiding unnecessary crawling jobs. But the OGC Web site is still an important seed because WMSs that are not located in the NASA and NOAA's reachable network may connect from other links in the OGC Web site.

Figure 12 was generated from the experimental results of crawling the first 40,000 web pages starting at NASA, NOAA, and OGC. In those results, 23 WMSs were found. The figure shows that both algorithms have similar trends because of the FIFO algorithm utilized. However, the proposed algorithm, which applies an ATF-based conditional probability

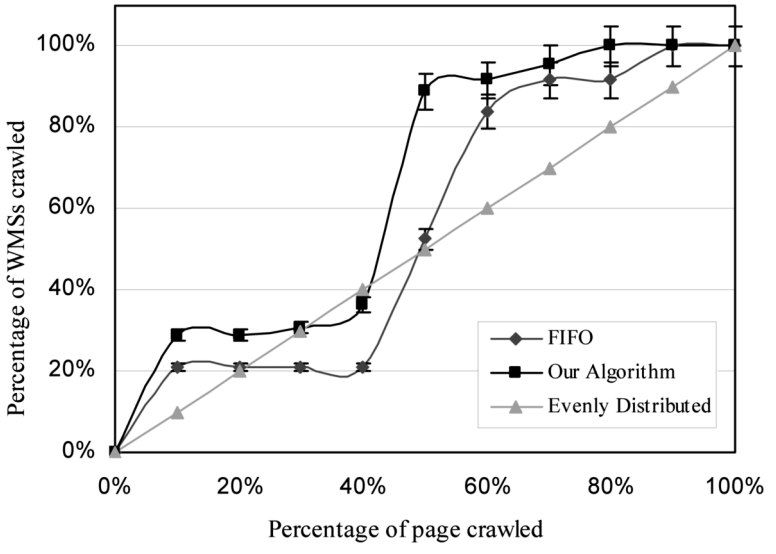

Figure 12. Quickness in finding WMSs by FIFO with and without ATF-based conditional probability model applied.

model to FIFO, is obviously faster than the pure FIFO algorithm. For example, when crawling 50% of the total pages, the proposed algorithm had found nearly 90% of the WMSs, while pure FIFO had only found 50% of those. After crawling 90% of the total pages, the proposed algorithm had found all of the WMSs, but the pure FIFO algorithm only found 90% of the WMSs. This is because the ATF-based conditional probability model is able to learn from past results and rank the frequencies of terms in the WMS vocabulary progressively. In this way, irrelevant portions of the Web are filtered out and the crawling scope is narrowed down, so that URLs that are more likely to be a WMS can be crawled earlier.

From Figure 12, we have an interesting finding in the distribution pattern of WMSs. The curves (for our algorithm and FIFO) go up quickly from 40% to 50% of crawled web pages (on the *x*-axis), which means that the crawler found a lot of WMSs (nearly 40% of all, as shown on the *y*-axis), but from 60% to 100% of crawled web pages (on the *x*-axis), the curve is steady and only 10% of total WMSs were found. We also conducted the experiment by crawling from other candidate seeds, such as (1) GeoPortal (http://geoportal.gc.ca/index_en.html), (2) the atlas of Canada (http://atlas.gc.ca), (3) German Aerospace Center (http://www.dlr.de/en/desktopdefault.aspx), and (4) British Geological Survey (http://www.bgs.ac.uk/). All the experiments showed similar trends of WMS distribution – they are *dispersed globally and clustered locally*. It is not hard to see that the WMSs are located in different countries, which are dispersed widely on the Web. While in a virtual place of the Web, the publishers/researchers for WMSs are usually limited to the geospatial domain and they usually link to other WMS publishers with similar topics. Therefore, a small-scale linkage graph for WMSs is formed. However, if WMSs were distributed evenly, the precision curve would be a straight line as depicted in Figure 12.

To furthermore validate this finding, we compared both the geographic and Web distributions of WMSs retrieved by our crawler and by the RR crawler. Results showed that the total 1126 (611 from GIDB and 515 from others) WMSs our crawler found are scattered in 18 countries (shown in Figure 13b, light gray and black bars), where the United

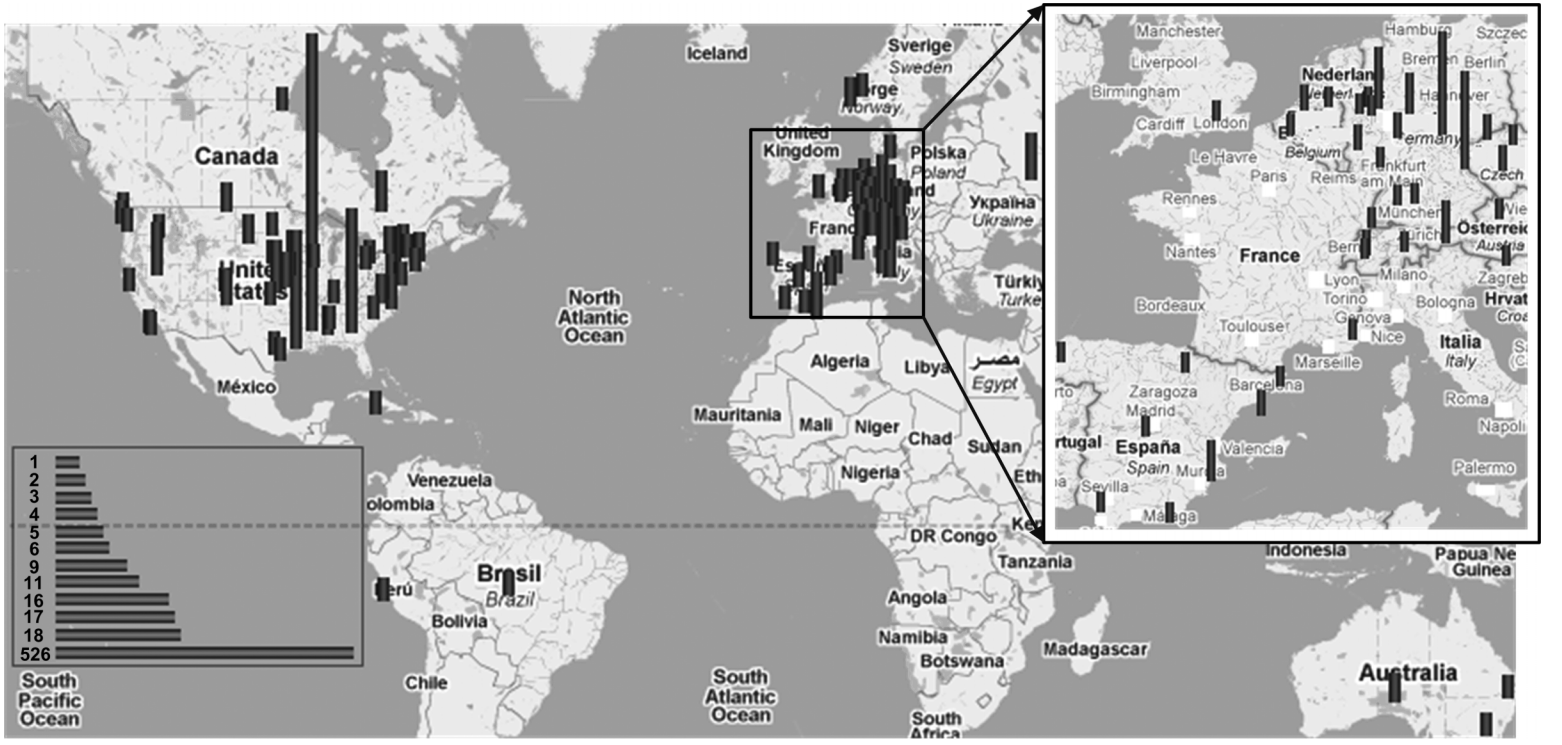

Figure 13. Comparison of global distribution of WMSs found by RR and our Crawler. All the bars use the same scale, shown in the legends of (a) and (b). The legend in (b) shows numbers that do not exist in the legend of (a). (To illustrate the results on the map clearly, the lengths of some of the bars are not to scale.) (a) Global distribution of WMSs found by RR (by December 2008). (b) Global distribution of WMSs found by our crawler (by December 2008 and June 2009). There are three types of bars, which are black, light gray, and dark gray. Black bars are the WMS distribution identified in December 2008. Light gray bars show the WMSs that were available in December 2008 but were unavailable in June 2009. Dark gray bars demonstrate the increased WMSs from December 2008 to June 2009.

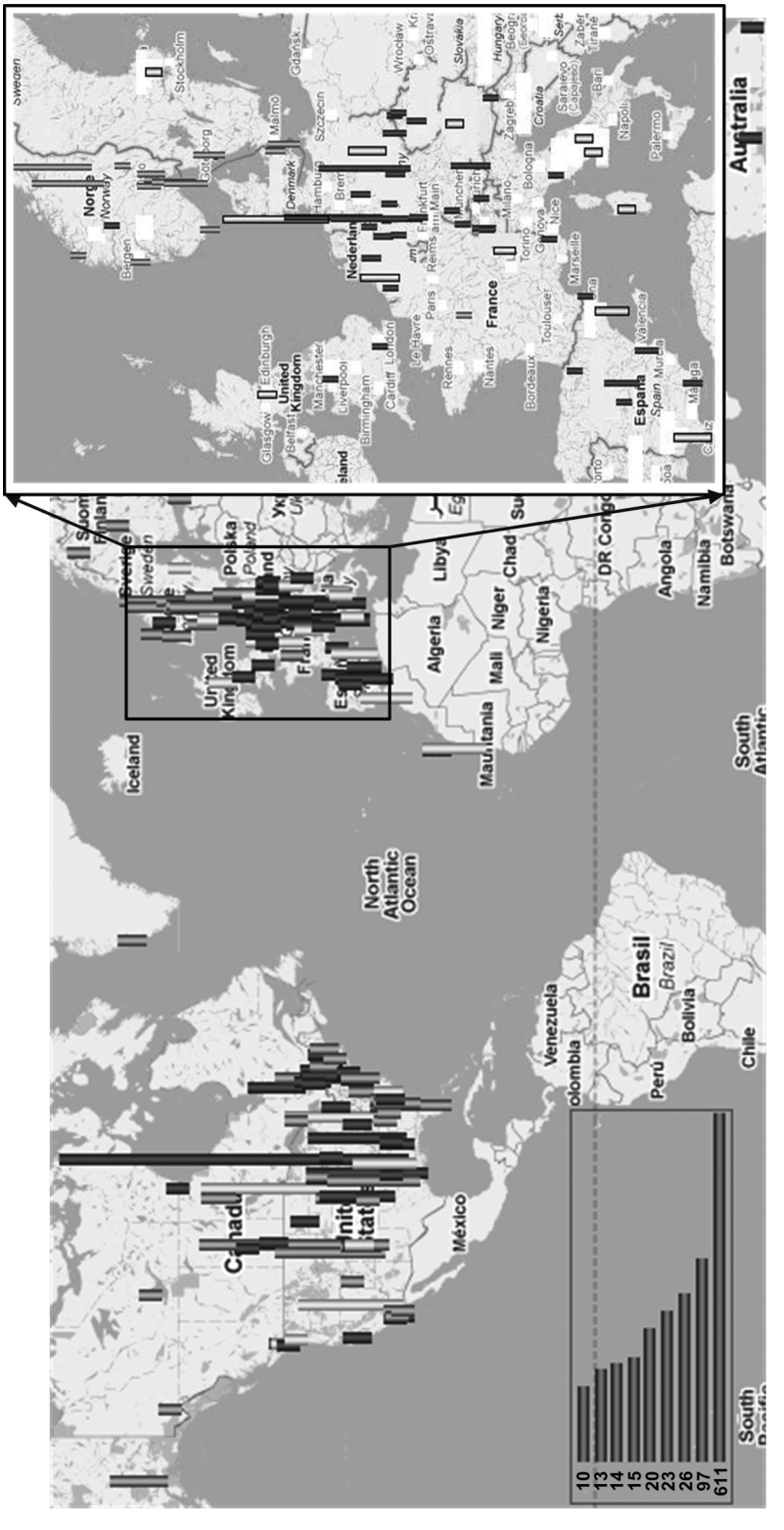

Figure 13. *(Continued)*

States hosts the most WMSs (852), then Germany (145), Canada (42), Spain (38), Netherlands (12), Switzerland (6), Czech Republic (6), Australia (5), United Kingdom (4) and others (17). The WMSs found by RR (Figure 13a) are located in 20 countries, mainly in United States, Germany, Canada, Spain, Czech Republic, Australia, Russian Federation, Netherlands, and others. Results from both crawlers illustrated the clustering of WMSs as well. On average, our crawler found 8 services per server, while the RR crawler found 6 services per server.

We also observed the Web diffusion of WMS servers over time (using a 6-month frame) to illustrate distribution variation. By June 2009, our crawler had identified 228 more live WMSs clustered on 68 web servers (dark gray bars in Figure 13b) than the number found by December 2008 (Figure 11). The United States had the biggest increase in absolute number (146). Services in Europe had increased by 78, which were distributed among Norway (60), Sweden (11), Denmark (5), France (1), and Finland (1). The crawler had also found 4 more WMSs in Canada. However, 168 WMSs (light gray bars in Figure 13b) identified by December 2008 became dead by June 2009. This phenomenon reflects the fact that the stability cycle of a WMS is short. Although more WMSs are becoming available to provide online geospatial data services, they still lack effective maintenance methods. Meanwhile, the frequent changes of WMS hosting makes it hard for current catalogs to adapt. Therefore, although crawling techniques proposed in this paper can automatically detect the changes, the instability of WMSs creates a very challenging issue within the geospatial community.

In total, the unique web servers hosting these WMSs increased to 186 and on average there were 6.38 services per web server. These results still show that WMSs are highly clustered. Note that although the crawling task has been conducted for a long time and a significant number of WMSs have been identified, there is still a big vacuum in Asian and African countries. In contrast, countries like the United States and Canada in North America and European countries such as Norway, Sweden, and Denmark all have significant contributions to the total WMSs. This may reflect the fact that the research of GWSs and interoperability in some Asian and African countries need great advancement.

Overall, the trends shown in Figure 13 demonstrate the clumped distribution of WMSs. This distribution pattern indicates that WMSs can be more effectively discovered by designing a focused and small-scale crawler than by relying on general search engines like Google. ‘Small scale’ is a relative measurement compared to the total size of the Web, estimated to be more than 17.27 billion (indexable) pages (Kunder 2009). In total, our crawler has crawled around 1.3 million web pages. In comparison to Google which uses more than 100,000 servers (Mingay 2007) to conduct crawling tasks, the design of the proposed crawler leads to a big savings in computing resources, network bandwidth, and crawling costs.

7. Discussion and conclusion

The increasing availability of the GWSs on the Internet poses a challenge in discovering GWSs from the vast amounts of information on the Internet. This paper reports on our research in improving crawling for quickly discovering GWSs. Multiple techniques, such as a priority queue and multi-threading, are adopted to improve the crawling process in the development of our crawler. The methodology, algorithm, and system architecture are described in detail. Performance evaluations are conducted from two aspects: (1) with and without optimization techniques and (2) comparing our crawler to the three other popular GWS crawlers. Results demonstrate that our crawler performs better than the others, in terms of efficiency and effectiveness, with the techniques adopted.

Although the proposed crawler has identified a significant number of live WMSs, we believe this number still underrepresents the total amount of the real-world WMSs. This is because there are a fair number of services residing in deep Web (Lawrence and Giles 1999), such as those behind firewalls or hosted on private servers (Ramsey 2005). Solving this problem needs the efforts from researchers in both GIS and information retrieval (IR) communities. Currently, GIS researchers propose to develop a supportive environment for crawlers by implementing a 'public library' through Universal Description Discovery and Integration (UDDI) (Reichardt 2005). In the IR community, alternative ways are being exploited to facilitate the crawling of the deep Web. For example, Google provides the Sitemap Protocol and mod oai mechanism to enable search engines and other interested parties to discover deep Web resources on particular web servers (Brandman *et al.* 2000). They accomplish this by allowing the servers to advertise the accessible URLs to search engines such that resources that have no obvious linkage can be discovered automatically. Another way is to exploit the deep Web using human crawlers as supplements to machine crawlers, which means that humans find interesting links that machine crawlers cannot find at the current stage in Web harvesting, e.g. StumbleUpon (http://www.stumbleupon.com). In addition, the process of seed selection discussed in Section 6.3 can be considered as human crawling, which assists the algorithmic crawler proposed. We believe that with optimization techniques proposed in this study and the maturing of the supportive environments, future crawling would be more efficient and effective.

Furthermore, the clumped distribution of WMSs was discovered: they are globally dispersed and locally clustered. Following this pattern, the OpenGIS community would be able to build a larger and more open environment for geospatial information sharing and interoperating. The above findings also provide effective guidance and principles for designing an efficient crawling engine to discover WMSs. As the WMSs are Web services in nature, the algorithms used for discovering WMSs are also applicable for discovering other types of Web services. In addition, with the significant number of services identified by the proposed crawler, semantic relationships can be analyzed based on the content described in the capability file of each service and a knowledge base can be built to allow semantic query from available services to enhance traditional search ability (Li *et al.* 2008).

In the future, we aim to integrate more effective politeness policies such as a robots exclusion protocol and a comprehensive fault-tolerant mechanism. We will also combine our research outcome with other advanced computing techniques, such as grid and peer-to-peer computing, to further improve the performance (Yang *et al*. 2008) of crawling and utilization of WMSs and other GWSs for application development (Yang *et al.* 2007).

Notes

1. Refractions Research (RR)'s white paper. http://refractions.net/expertise/whitepapers/ogcsurvey/ogcsurvey/
2. Skylab Mobilesystems' WMS list. http://ogc-services.net/
3. GIDB WMS Service list: http://columbo.nrlssc.navy.mil/ogcwms/servlet/WMSServlet?REQUEST=ServiceLinks

References

Alameh, N., 2003. Chaining geographic information Web services. *Internet Computing*, 7, 22–29.

Al-Masri, E. and Mahmoud, Q.H., 2007. Interoperability among service registry standards. *Internet Computing*, 11, 74–77.

Brandman, O., *et al.*, 2000. Crawler-friendly web servers. *SIGMETRICS Performance Evaluation Review*, 28, 9–14.

De La Beaujardiere, J., ed., 2004. *Web Map Service implementation specifications, Version 1.3*. OGC Document Number: 04-024, Open Geospatial Consortium, 85pp. Available online at: http://portal.opengeospatial.org/files/?artifact_id=5316 (accessed 15th January 2010).

Dogac, A., Kabak, Y., and Laleci, G.B., 2006. *ebXML Registry Profile for Web Ontology Language (OWL)*. Organization for the Advancement of Structured Information Standards (OASIS), 75pp. Available online at: http://xml.coverpages.org/OASIS-regrepOWL-Profile-22611.pdf (accessed 15th January 2010).

Dill, S., *et al.*, 2002. Self-similarity in the Web. *ACM Transactions on Internet Technology*, 2, 205–223.

Egenhofer, M., 2002. Toward the semantic geospatial Web. *In*: *Proceedings of the 10th ACM international symposium on advances in geographic information systems*, VA, A. Voisard and S. Chen (Eds.) New York: ACM, 1–4.

Fesenmaier, D.R., Wober, K.W., and Werthner, H., 2006. *Destination recommendation systems: behavioral foundations and applications*. Oxford, UK: Oxford University Press, 205–220.

Keller, U., Lara, R., and Polleres, A., 2004. WSMO Web service discovery. *In*: U. Keller, *et al.*, eds. *WSMO Web service discovery working draft D5.1v0.1*, 13–15. Galway, Ireland: DERI. Available online at: http://www.wsmo.org/2004/d5/d5.1/v0.1/ (accessed 15th January 2010).

Kunder, M.D., 2009. *The size of World Wide Web* [online]. Available from: http://www.worldwidewebsize.com/ [Accessed September 2009].

Lawrence, S. and Giles, C.L., 1999. Accessibility of information on the Web. *Nature*, 400, 107–109.

Li, W., 2007. *Interoperability based spatial information retrieval and ontology based semantic search*. Thesis (MSc), Institute of Remote Sensing Applications, Chinese Academy of Sciences, Beijing, China, 84pp.

Li, W., Yang, C., and Raskin, R., 2008. A semantic enhanced search for spatial Web portals. *In*: *Technical report of semantic scientific knowledge integration of AAAI spring symposium*, D. McGuinness, P. Fox, and B. Brodaric (Eds.). Merlo Park, CA: AAAI Press, 47–50.

Ma, X., *et al.*, 2006. A peer-to-peer approach to geospatial Web service discovery. *In*: *Proceedings of 1st international conference on Scalable information system*. Hong Kong, China, Article. 53. New York: ACM.

Mingay, S., 2007. Green IT: the new industry shock wave. *Gartner RAS core research note G00153703*, 2, Article 10. Gartner Inc: Stamford, Connecticut, US. Available online at: http://www.netdesign.dk/manedens-tema/telepresence/green-it-the-new-industry.pdf (accessed 15th January 2010).

NASA, 2007. *Earth system science data resources: tapping into a wealth of data, information, and services*. National Aeronautics and Space Administration, 65pp. Available online at: http://outreach.eos.nasa.gov/outreach/broch/daac/ESSDR112007.pdf (accessed 15th January 2010).

Nebert, D., ed., 2004. *Developing spatial data infrastructures: the SDI cookbook*, Version 2.0, Global Spatial Data Infrastructure, 171pp. Available online at: http://www.gsdi.org/docs2004/Cookbook/cookbookV2.0.pdf (accessed 15th January 2010).

Paul, M. and Ghosh, S.K., 2006. An approach for service oriented discovery and retrieval of spatial data. *In*: *Proceedings of the 2006 international workshop on service-oriented software engineering*, E. Di Nitto, R. J. Hall, J. Han, Y. Han, A. Polini, K. Sandkuhl, A. Zisman (Eds.), 88–94. New York: ACM.

Peytchev, E. and Claramunt, C., 2001. Inelegant transportation systems and network algorithms: Experiences in building decision support systems for traffic and transportation GIS. *In*: *Proceedings of the 9th ACM international symposium on advances in geographic information systems GIS′01*, GA, 154–159. W. G. Aref (Ed.). New York: ACM.

Rae-Dupree, J., 2006. Dash navigation – real-time GPS driving. *Business Journal*, Available from: http://sanjose.bizjournals.com/sanjose/stories/2006/11/20/focus15.html [Accessed 23 January 2008].

Rajasekaran, P., *et al.*, 2004. Enhancing Web services description and discovery to facilitate composition. *In*: *Proceedings of first international semantic Web services and Web process composition workshop*. San Diego, CA J. Cardoso and A. Sheth (Eds.) Berlin: Springer–Verlag.

Ramsey, P., 2005. A census of public OGC Web services. Presentation to the OGC planning 600 Q19 committee, 2005 CGDI Implementor's Forum. Available online at: http://www.geoconnections.org/developersCorner/devCorner_devNetwork/meetings/2005.02.16/cgdi_200502_ows_survey.ppt (accessed 15th January 2010).

Ran, S., 2004. A model for Web services discovery with QoS. *SIGEcom Exchanges*, 4, 1–10.
Rauschert, I., *et al.*, 2002. User interfaces: designing a human-centered, multimodal GIS interface to support emergency management. *In*: *Proceedings of the 10th ACM international symposium on advances in geographic information systems GIS '02*. VA, 119–124. A. Voisard and S. Chen (Eds.), New York: ACM.
Reichardt, M., 2005. GSDI depends on widespread adoption of OGC standards. *In*: *Proceedings of the FIG working week and GSDI8: from pharaohs to geoinformatics*. Cairo, Egypt.
Roman, D., *et al.*, 2005. Web service modeling ontology. *Applied Ontology*, 1, 77–106.
Sample, J.T., *et al.*, 2006. Enhancing the US Navy's GIDB portal with Web services. *Internet Computing*, 10, 53–60.
Schutzberg, A., 2006. Skylab Mobilesystems crawls the Web for Web Map Services. *OGC User*, 8, 1–3.
Sergey, B. and Page, L., 1999. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30, 107–117.
Singh, G., *et al.*, 2003. A metadata catalog service for data intensive applications. *In*: *Proceedings of the 2003 ACM/IEEE conference on supercomputing*. Washington, DC: IEEE Computer Society.
Steele, R., 2001. Techniques for specialized search engines. *In*: *Proceedings of the international conference on Internet computing, IC'2001*. Las Vegas, NV, 25–28. P. Graham, M. Maheswaran and M. R. Eskicioglu (Eds.). Las Vegas, Nevada: CSREA Press.
Stevens, D., Dragicevic, S., and Rothley, K., 2007. iCity: a GIS-CA modeling tool for urban planning and decision making. *Environmental Modeling and Software*, 22, 761–773.
Stock, K., ed., 2009. *OGC catalogue services – OWL application profile of CSW*. OGC document no. 09-010, Open Geospatial Consortium, 70pp.
Vretanos, P.A., ed., 2005. *Web feature service implementation specifications, version 1.1.0*. OGC document no. 04-094. Open Geospatial Consortium, 131pp.
Whiteside, A. and Evans, J., eds., 2006. *Web coverage service implementation specifications, version 1.1.0*. OGC document no. 06–083r8. Open Geospatial Consortium, 143pp.
Wöber, K., 2006. Domain specific search engines. *In*: D.R. Fesenmaier, K. Wöber, and H. Werthner, eds. *Destination recommendation systems: behavioral foundations and applications*, Wallingford, UK: CABI, 205–226.
Yang, C., Cao, Y., and Evans, J., 2007. WMS performance and client design principles. *Journal of GIScience and Remote Sensing*, 44, 320–333.
Yang, C. and Tao, C.V., 2006. Distributed Geospatial Information Service (Distributed GIService). *In*: S. Rana and J. Sharma, eds. *Frontiers of geographic information technology*. New York: Springer, 103–120.
Yang, C., *et al.*, 2005. Performance improving techniques in WebGIS. *International Journal of Geographical Information Science*, 19, 319–342.
Yang, C., *et al.*, 2008. Distributed geospatial information processing – sharing distributed geospatial resources to support the digital Earth. *International Journal of Digital Earth*, 1, 259–278.
Yue, P., *et al.*, 2007. Semantics-based automatic composition of geospatial Web service chains. *Computers and Geosciences*, 33, 649–665.