Computers, Environment and Urban Systems 59 (2016) 195–207

Contents lists available at ScienceDirect

Computers, Environment and Urban Systems

journal homepage: www.elsevier.com/locate/ceus

PolarHub: A large-scale web crawling engine for OGC service discovery in cyberinfrastructure

Wenwen Li *, Sizhe Wang, Vidit Bhatia

School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ 85287-5302, United States

ARTICLE INFO

Article history:
Received 4 March 2016
Received in revised form 14 July 2016
Accepted 16 July 2016
Available online 20 July 2016

Keywords:
PolarHub
Big data access
Geospatial interoperability
Scalability
Cyberinfrastructure

ABSTRACT

The advancement of geospatial interoperability research has fostered the proliferation of geospatial resources that are shared and made publicly available on the Web. However, their increasingly availability has made the identification of the web signature of voluminous geospatial resources a major challenge. In this paper, we introduce our solution of a new cyberinfrastructure platform, the PolarHub, that conducts large-scale web crawling to discover distributed geospatial data and service resources and accomplish this goal efficiently and effectively. The PolarHub is built-upon a service-oriented architecture (SOA) and adopts Data Access Object (DAO)-based software design pattern to ensure the extendibility of the software system. The proposed meta-search-based seed selection and pattern-matching based crawling strategy facilitates the rapid resource identification and discovery through constraining the search scope on the Web. In addition, PolarHub introduces the use of advanced asynchronous communication strategy, which combines client-pull and server-push to ensure high efficiency of the crawling system. These unique design features of PolarHub enable a high performance, scalable, sustainable, collaborative, and interactive platform for active geospatial data discovery. Because of OGC's widespread adoption, OGC-compliant web services become the primary search target of PolarHub. Currently, the PolarHub system is up and running and is serving various scientific community that demands geospatial data. We consider PolarHub a significant contribution to the field of information retrieval and geospatial interoperability.

1. Introduction

21 century is the era of big data. Benefiting from the advancement of data collection techniques, a wide spectrum of in-situ sensor networks and Earth Observing Systems have been deployed or launched to continuously monitor the Earth from multi-angles and multiple dimensions (Di, Chen, Yang, & Zhao, 2003; Hart & Martinez, 2006; Meier, Marquis, Kaminski, & Weaver, 2004). As a result, a huge amount of spatial data has been produced, archived and disseminated to benefit the broader scientific communities, ranging from ecology (Ustin et al., 1991) to hydrology (Kawanishi et al., 2003), from polar science (Manney et al., 2006) to climate studies (Asrar & Dozier, 1994), from disaster management (Kääb et al., 2003) to national security (Busby, 2008), to energy (Kawanishi et al., 2003) and land-water nexus (Steffen, Jager, Carson, & Bradshaw, 2002).

To maximize the values of voluminous geospatial data to support scientific research, the geospatial community has been developing standards and spatial data infrastructures to enable the widespread sharing and interoperability of the available dataset. Geospatial web services are a set of such services developed and deployed in a service-oriented architecture (Lake & Farley, 2007) to provide data or geoprocessing in a standardized way. This communication method eliminates some obstacles in the exchange, reuse, interoperation and integration among heterogeneous resources, and significantly improves geospatial interoperability among distributed earth observation systems. The Open Geospatial Consortium (OGC), a non-profit organization comprised of almost 500 industry partners, government agencies and universities, pioneers and standardizes new web service specifications for accessing publicly available geospatial information. Popular web service standards include: Web Map Services (WMS) that deliver data as static images, Web Feature Services (WFS) that provide data as an encoded feature set, or Web Coverage Services (WCS) that return actual continuous raster data, to mention a few. Due to their applicability, there has been a massive proliferation and widespread adoption of OGC web services (OWS) in various earth science applications over the past decade, ranging from disaster management (Weiser & Zipf, 2007), to environmental modeling (Granell, Díaz, & Gould, 2010); from urban economic simulation (Li et al. 2013), to building smart social software (Klamma, Cao, & Spaniol, 2008). The OWS nowadays is also a key component that enables the national and international spatial data infrastructure entities (Bernard, Kanellopoulos, Annoni, & Smits, 2005; Khalsa, Nativi, & Geller, 2009; Nebert, 2008; Rautenbach, Coetzee, & Iwaniak, 2013) and various "big-iron" cyberinfrastructure portals (Li et al.,

* Corresponding author.
E-mail address: Wenwen@asu.edu (W. Li).

http://dx.doi.org/10.1016/j.compenvurbsys.2016.07.004
0198-9715/

2011; Wang, 2010; Williams et al., 2009; Yang, Raskin, Goodchild, & Gahegan, 2010) to advance research frontiers in earth sciences.

However, the decentralized nature of the geospatial web services has made the resource discovery an increasingly challenging task. First, geospatial data/services is often "Deep Web" data that resides in community-specific databases and has different web signatures than "Surface Web" data (Malik & Foster, 2012). Second, although the number of available geospatial services is expanding at a very fast pace, it only occupies a small portion in comparison to the total volume of the Web. Distilling such information from massive Web data of diverse types becomes an extremely difficult task. Third, most scientific analysis is conducted by long-tail researchers, for whom data discovery may become a nightmare because they do not have an accessible tool available that can rapidly locate needed data on the Internet (Heidorn, 2008).

Fortunately, increasing attention to this problem has driven efforts to create intelligent connections between the data consumers and the geospatial services hiding on the Internet. For instance, government agencies such as the National Aeronautics and Space Administration (NASA), the National Oceanic and Atmospheric Administration (NOAA), and United State Geological Survey (USGS) have established online data catalogs at their physically distributed data centers to support registration, search and harvest of geospatial data and services. Exemplary works including the data.gov site established by the US General Services Administration to openly share vast amounts of government data (Lakhani, Austin, & Yi, 2010) and the Global Earth Observation System of Systems – GEOSS clearinghouse (Liu et al., 2011) which links together the data systems from various Earth Observation programs. The National Science Foundation has also started to investigate such cyberinfrastructure and data science initiative. The EarthCube program was established in 2011 to develop community-driven solutions to improve our understanding of Earth as a complex system. Very recently, the DataONE team was awarded 15 million to establish a distributed cyberinfrastructure the meet the need of scientists and decision makers for the "open, persistent, robust, and accessible Earth observational data" (DataONE, 2014). Both programs have considered the goal of discovering, organizing and providing easy access to distributed geospatial resources a top priority. Techniques such as service brokering (Nativi, Craglia, & Pearlman, 2013) and linked open data (Janowicz, Scheider, Pehle, & Hart, 2012) were proposed to improve data interconnection, increase their visibility, and make them more discoverable. However, these programs are still in their infancy, and developing an efficient, scalable, ready-to-use tool to enable the automatic discovery of distributed geospatial data and services is still an open problem.

In this paper, we report our effort in building a large-scale web crawling engine, the PolarHub, to actively mine the Web to discover distributed geospatial data resources. Because of its widespread adoption, the OGC-compliant web services become the primary search target of PolarHub. Specifically, we aim to identify a Web endpoint represented by a Universal Resource Locator (URL) u, which has the following characteristics. First, u is dynamic rather than being static. It should be able to accept a request with specific parameter patterns $=\{p_1, p_2, \ldots p_n\}$, in which p_i is a parameter pattern that meets the requirement of a OGC web service specification, such as WMS. It should also respond in a reasonable time frame and return $d=\{d_1, d_2, \ldots d_n\}$, where d_i is either a structured XML (Extensible Markup Language) document, describing the metadata of the dataset that u is providing, or it could be a binary dataset depicting the geospatial information of the requested data layers. Only the URLs that satisfy both constraints $\langle p, d \rangle$ will be considered as our search target.

The rest of the paper is organized as follows: we first review earlier works on service discovery in Section 2, introduce in detail the service-oriented system architecture in Section 3, describe the crawling algorithm and other advanced techniques in Section 4, demonstrate the Graphic User Interface and its functionality in Section 5, demonstrate the impact of crawling performance in terms of its parameters in Section 6, and summarize and conclude this work in Section 7.

2. Literature

Data/service discovery is recognized as one of the most important research topics in many science disciplines, because of its data-driven nature and because tons of physical and social applications require the use of adequate data to produce new knowledge (Farmer & Pozdnoukhov, 2012; Li, Goodchild, & Raskin, 2014; Sui & Goodchild, 2011). The importance of service discovery is emphasized in Foster (2005), as all of these "[Web] services have little value if others cannot discover, access, and make sense of them."

Earlier approaches for discovering web services rely on a protocol named UDDI (Universal Description, Discovery, and Integration). This protocol derives from an industry initiative to classify, catalog and manage web services using a XML (eXtensible Markup Language)-based catalog. The information registered in UDDI can be classified into three layers: whitepage, yellowpage and greenpage. Whitepage provides information about a business entity that provides a web service, Yellowpage lists the category of service providers based on standard taxonomy, and Greenpage describes the technical information about a web service, encoded usually in a WSDL (Web Service Description Language). This service registry model is complicated, requires too much data for its purpose, and has limited use only in industries. Rather than fostering human computer interaction, UDDI focuses more on the machine-to-machine interaction for service registration and service search. Due to these reasons, UDDI has not received widespread adoption as a service registry standard.

In earth science domain, centralized catalogs have become a major means for publishing and discovering earth observation data and services (Kottman, 1999). Existing catalogs, such as Esri's Geoportal server (http://gptogc.esri.com/geoportal/), data.gov (http://www.data.gov/), Spatineo Directory (http://www.spatineo.com/) and others adopt the concept of 'metadata' – the data about data, to describe the content, functions, and the online endpoints of available geospatial resources in a standardized manner (FGDC, 1998ISO, 2003). Metadata records are parsed and stored according to information models implemented by different catalogs. For instance, OGC Catalog Service for the Web (Nebert & Whiteside, 2004), or CSW, is a catalog service that provides indexing and searching capability by implementing the ebXML Registry Information Model. To allow remote service discovery, the CSW catalog provides a web interface to allow end users to locate service of interest through keyword search. At the same time, a number of information protocols, such as ANSI Z39.50 (ANSI, 1995), OpenSearch (LeVan, 2013), and OAI-PMH (Van de Sompel & Lagoze, 2001), are supported in addition to the CSW standard to facilitate cross-catalog harvesting. Other catalog standards include Environmental Distributed Data Services (THREDDS) and (Open source Project for a Network Data Access Protocol) OPeNDAP, but they are more often used to provide scientific datasets rather than services.

The catalog solutions are effective in facilitating remote data discovery. However, the coverage of data services residing in the catalog depends heavily on manual registration by the service providers. There lacks an automatic mechanism for these catalogs to actively search for available services (Li et al., 2011). Meanwhile, the catalogs are distributed as well: a global service repository remains missing.

To overcome this limitation in existing catalogs, researchers from universities, government agencies, and the private sector began to investigate ways to automate the service discovery process (Cho & Garcia-Molina, 2002; de Andrade, de Souza Baptista, & Davis, 2014; Li, Yang, & Yang, 2010; Lopez-Pellicer et al., 2011; Sample, 2006; Schutzberg, 2006). Notable solutions include the use of a web crawler. Crawlers start from one or more seed webpages and analyze the source page to extract all outgoing hyperlinks. Then these links are visited and the same extraction process is performed on them. In this way, the whole Web will be visited and relevant information can be extracted from each of the webpages being visited. Li et al. (2010) developed such a crawler for automated searching of OGC WMSs on the Internet.

It adopts a conditional probability model to determine the probability that a webpage contains a WMS endpoint. This narrows the search scope and rapidly identifies services. This crawler is a stand-alone application and is not supported by other search engines. Lopez-Pellicer et al. (2011) uses Google search API to extract OGC Web Services from Google search results. Though this is information extraction, not a real crawling strategy, an over eight-thousand services were discovered. Certainly, the number of matched records (less than 4000) Google returns limits this technique's generality.

In this paper, we discuss the design and implementation of a new web crawling tool – the PolarHub, a tool capable of conducting large-scale search and crawling of distributed geospatial data from the broad cyberspace. Large-scale means PolarHub is not limited to search within a certain geographical boundary or a certain portion of the cyberspace. By initiating crawling from a variety of websites and web hosts, PolarHub is looking for data at a global extent. The next sections discuss the technical innovation in detail.

3. System architecture of PolarHub

Fig. 1 demonstrates the general component architecture of the PolarHub crawler, which is organized into three main software tiers. The "Crawler Client" tier provides functionality related to management of the crawling tasks, visualization and statistics of the crawling results. Additionally, this layer provides functionality like user management, result search, and download. Different from other crawler implementations (de Andrade et al., 2014; Li et al., 2010; Lopez-Pellicer et al., 2011; Sample, 2006; Schutzberg, 2006) the crawling process of which are always sequential, PolarHub is able to support concurrent crawling on multiple tasks created by multiple users. This feature is enabled by a new multi-layer thread pool concurrency model residing in the crawler "Middleware" layer. The top-level thread pool is termed a controller pool, which maintains a number of threads to carry out concurrent crawling tasks. These controller threads are managed by a main thread that receives queued tasks from the client and allocates an idle thread worker in the pool to execute the task. One controller thread usually manages multiple crawler threads in the crawler thread pool to achieve high performance crawling. We classify the crawler threads into general crawlers ("G" in Fig. 1) and dedicated crawler ("D" in Fig. 1), each in charge of different crawling tasks. The general-purpose crawler takes care of large-scale web crawling according to user's requirement on service content and service type. In contrast, the dedicated crawler establishes a dedicated communication channel to connect with data centers that do not provide standard service retrieval protocol, such as CSW. The design and implementation of these crawlers are discussed in detail in Section 4. When the crawling process ends or other tasks have been successfully executed, the results are returned to the controller thread. It then disseminates information appropriate to a user's privileges to the front-end user interface. The crawler pool retains a connection with backend database, for saving crawling results and status for a real-time update to the users. After a crawler finished crawling, its controller thread terminates and is recycled by the main thread to assign to other tasks.

The advantages of this multi-layer thread pool model lie in three strengths. First, it avoids thread spawning and reduces overhead from the thread-per-request model, in which a new thread is created every time a request arrives, exhausting system resources fairly quickly. Second, it allows high performance crawling since multiple crawler agents can be concurrently started for the same task. Third, it allows easy deployment and scale-out of the crawling model onto a distributed system through message passing strategy with each controller thread and its corresponding crawler agents running on different servers in a cluster.

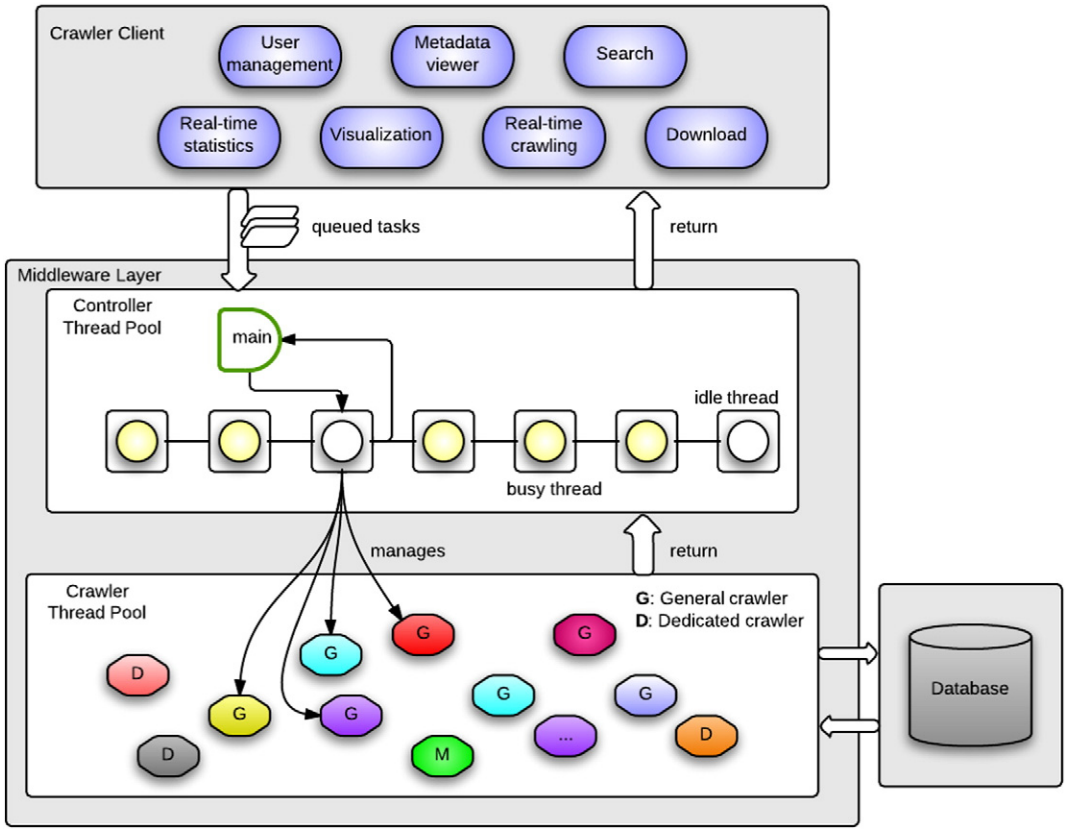


Fig. 1. PolarHub general components architecture.

4. Methodology

4.1. PolarHub class design diagram

The internal structure of the web crawler is demonstrated in the UML class diagram of Fig. 2. The classes grouped in white achieves general-purpose web crawling, which starts from a seed URL, accesses the source webpage of this URL, then extracts and puts all hyperlinks (URLs) in a first-in-first-out queue (*Q*) from this webpage, and then keeps accessing the URLs until the queue becomes empty. The class *CrawlController* is the access point for any crawling tasks, conducted by *WebCrawler* agents it starts. All crawl controllers are maintained in the controller thread pool (Fig. 1 top layer in the "Middleware Tier") and managed by the system, whereas all crawler agents (Fig. 1 bottom layer in the "Middleware Tier") are managed by a *CrawlControler* instance.

Each *CrawlController* has *Frontier* and *DocIdServer* properties. The former class operates the crawling queue *Q* and is responsible for scheduling a URL or a set of URLs to visit. *Frontier* defines another function *getNextURLs()*, which dequeues the scheduled URLs from *Q* and assigns it/them to a crawler agent. The latter class *DocIDServer* manages the webpages that have been visited. The data structure *Q* and the class *DocIDServer* are coordinated to avoid duplication in crawling, which is an important concern in designing concurrent crawlers. The specific mechanism is: crawling queue *Q*, which maintains all the URLs to be crawled, does not allow duplicated entries. This means, if a URL found by a crawler worker has already existed in *Q*, this URL cannot be added to *Q* a second time. At the same time, the URL of each webpage that has been visited will be recorded in PolarHub's backend database. Each such URL will be assigned a unique document ID through function *getNewDocID()* defined in the class *DocIdServer*. If the *getNewDocID()* of a recently found URL has existed in the database, this URL has been visited and will not be crawled again.

CrawlControler has three properties, all related to running crawler agents in the *WebCrawler* class. A controller can start any number of crawler agents by invoking the *start()* function. It can also assign more tasks to the crawlers by adding new seeds (a URL entry point to start crawling the Web) or shutdown a crawler agent when there are no more tasks to be assigned. When a crawler agent is started, it receives two objects of the class *Frontier* and *DocIDServer* and an object of the *CrawlController* itself. The crawler agent has another property, *Parser*, in charge of extracting the content of a webpage. This *Parser* instance is invoked by the *processPage()* function, which extracts the URLs from a webpage and schedule them to be processed through *Frontier*.

These main classes are sufficient for general-purpose crawling and will visit the entire Web starting from an entry point, known as a seed URL, and then can gradually access all other webpages through accessing the outgoing hyperlinks, or sub-URLs, from the seed URL and its sub-URLs. When it comes to searching for geospatial data in OGC service format, rules to narrow the scope of crawling and determine the existence of these services on the webpages are needed. The general-purpose crawler also needs to be tailored to record real-time crawling information. To accommodate this, two subclasses *PolarHubCrawler* and *PolarHubCrawlerControler* are defined (dark grey classes in Fig. 2), inheriting the *WebCrawler* class and the *CrawlController* classes respectively. The *PolarHubCrawler* defines three filters to determine whether a URL should be visited or not. First filter helps to remove URLs referring to a style sheet, a video clip, or a compressed file that matches a pattern defining these file extensions. The second filter removes social media websites from being considered

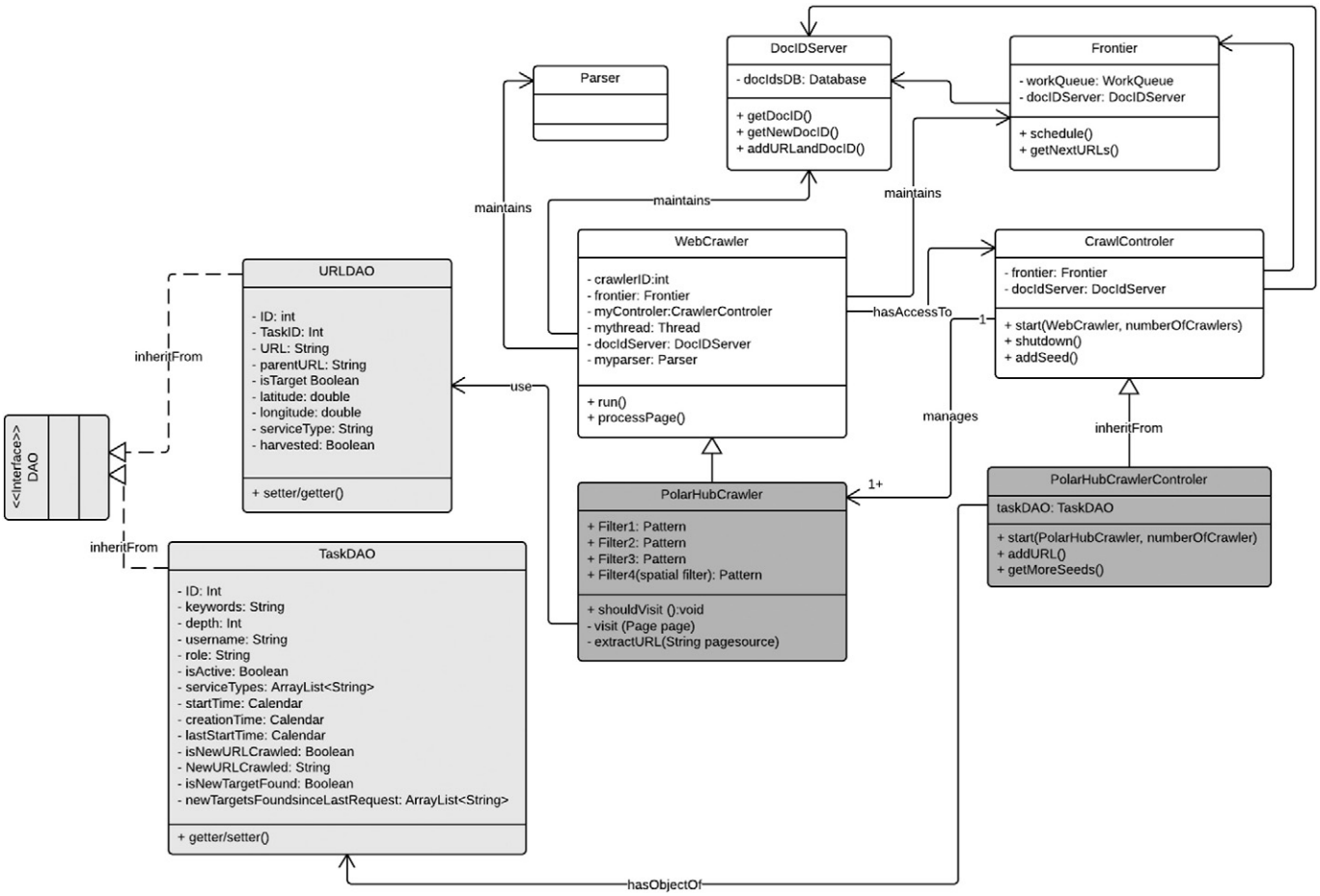


Fig. 2. PolarHub class design diagram.

because a crawler agent may easily get trapped in these websites without finding resources. The third filter is a spatial filter, which rejects a target even if the resource identified does not provide data within the required geographical boundary. The third filter is optional as it is equivalent to performing a spatial query after all target services are crawled. The *visit()* function defined in class *PolarHubCrawler* is the core function, as it defines the extraction strategy for the near-target resources and decides when to add new seed URLs to the crawling queue during the crawling process. A detailed description of the crawling strategy and algorithm is discussed in Section 4.2.

In this framework, the Data Access Object (DAO) pattern used. DAO is desirable because it separates persistent data storage, kept in the database, from the business logic, kept in the web application layer. This design pattern makes sure that modules within the framework are only loosely coupled, making the system extensible. Two DAO classes *URLDAO* and *TaskDAO* are defined. *URLDAO* records all information related to a URL visited by a crawler agent. This information includes: the ID of the task within the scope of which this URL is crawled (*TaskID*), its URL, the URL of a webpage from which this current URL is extracted (*parentURL*), whether this URL is a target (*isTarget*), its geolocation (*Latitude*, *Longitude*), service type (*serviceType*), which could be either one of the eight OGC service types PolarHub supports.

4.2. An effective crawling algorithm

Fig. 3 describes algorithm developed for the general-purpose crawling. The input parameters include crawling depth, crawling width, list of desired service types and a queue of URLs to be visited. Crawling depth is the distance between a target webpage and the seed webpage. This distance is measured by the minimum number of clicks from the seed webpage to visit the target webpage. The seed page has a depth of 0, and other webpages that have distances of 1 or 2 from the seed page will have a crawling depth of the same value. Crawler width determines the scope of the Web to exploit. It is an important factor used at the first phase of the crawling process: seed selection, the process of which is to determine which webpages we will start from to traverse the Web.

Because the Web contains clusters of webpages that are interconnected, the selection of crawling seeds from different clusters will improve the coverage of crawling. This can be considered as a sampling process, which is supported by a meta-search strategy. Namely, we conducted a search on Google and other popular general search engines and save the ranked results – the URLs of the webpages as the crawling seeds. The reason why the general search engines are used is because they maintain a huge amount of indexed webpages and utilize advanced ranking algorithms, i.e. PageRank (Langville & Meyer, 2011), to sort the webpages based on their popularity. Using this meta-search strategy, we were able to obtain a list of seed URLs ranked by its relevancy to a theme and this strategy avoids aimless crawling. These obtained URLs are first put into a first-in-first-out array *LoU* and then partitioned into different segments, each of which is assigned a width number *w*. The larger *w* is, the larger portion of the Web is going to be crawled. For instance, if 87 webpages are considered relevant to a keyword-based query, 87 is the length of *LoU* to be segmented. In our experiments, we set *k* as 10, this means the first 10 most relevant seed webpages are grouped together and they are given a crawling width of 1. Then the next 10 are given a crawling width of 2, and so on.

Besides crawling depth and width, another input parameter is the list of desired web service types of interest. PolarHub currently supports eight types of web services, including WMS, WFS, WPS, WCS, CSW, SOS, WMTS and WCPS. A description of these standards is listed in Table 1. A URL queue *q* is also provided to record all URLs to be visited. This is

```
Input: crawling depth: d
crawling width: w
desired service TYPE patterns: serviceTypePatt
desired service URL patterns: urlPatt
crawling URL queue: q
Output: a list l recording all recently found service URLs
1  if q is empty then
2      if crawling width i has not reached desired width w then
3          add to q all seeds ranked relevant by Google's PageRank mechanism on width i
4          i ←i + 1
5      else
6          crawler task finished, return
7  deQueue a URL u from q
8  if u matches the service URL pattern urlPatt and u is a live service then
9      determine and set service type of u by matching the service type patterns
       serviceTypePatt defined in Table 1
10     georeference u to obtain (latitude,longitude) based on u's IP address
11     push u to l
12 else
13     reconstruct a new URL u' from u based on service URL patterns urlPatt
14     add u' to list q
15 if u's crawling depth di less than d then
16     hl ← extract all URLs residing as hyperlinks on the webpage p of u
17     ul ← analyze p, extract existence of URLs as text
18     di ←di + 1
19     enQueue hl and ul to q
20 set u's status to be 'crawled'
```

Fig. 3. Pseudocode of PolarHub crawling algorithm.

Table 1
Regular expression for pattern matching and service identification.

Service type	Full name	Regular expression
WMS	Web Map Services	wms(?:t_ms|s)_capabilities
WFS	Web Feature Services	wfs[:_](?:(?:wfs_|)capabilities|featurecollection)
WPS	Web Processing Services	wps:(?:wps_|)capabilities
WCS	Web Coverage Services	wcs[:_]capabilities
CSW	Catalog Service for the Web	csw[:_]capabilities
SOS	Sensor Observation Service	<ows:servicetype[^>]*>.*sos.*</ows:servicetype>
WMTS	Web Map Tile Service	<ows:servicetype[^>]*>.*wmts.*</ows:servicetype>
WCPS	Web Coverage Processing Service	<ows:servicetype[^>]*>.*wcps.*</ows:servicetype>

maintained by the crawling controller and can be accessed by all crawler agents. A lock is put on *q* for mutual exclusion such that no two agents can alter it at the same time. Initially, a crawler agent will check the size of *q*. If it is empty and the crawling width *i* has reached the preset maximum value, the individual crawling task is finished and the crawling agents quit the process (Lines 5–6 in Fig. 4). If not, a new group of seeds will be added to *q* for further crawling (Lines 2–4 in Fig. 4).

When the *q* is not empty, the crawler agent will select a URL *u* at the front of *q* and start analyzing it. The analysis consists of two parts: web service identification (Lines 8–14 in Fig. 3) and URL extraction (Lines 15–19 in Fig. 3). To identify if *u* is referring to a service endpoint, a pattern-matching approach is adopted because typical OGC web services use formatted XML to encode the service metadata. According to the specificity of service standards, we adopt tag-scanning approach to make distinction among different services. For WMS, WFS, WPS, WCS and CSW, the contents of the root tag are checked and for the remaining three types, sub-tags with name ServiceType are checked. Table 1 lists the patterns in the content of each service type. These patterns are described in regular expressions, due to their speed, accuracy, and support for automatic pattern matching. Because a URL *u* that belongs to our search targets could be presented in multiple forms, we need to first inspect the URLs. For the URL that already contains a service endpoint with expected data/metadata request pattern and it is a live service/link (Line 8 in Fig. 3), the web content of the web page referred to by the URL will be further examined to determine the service type by matching the patterns defined in Table 1 (Line 9 in Fig. 3). However, when a URL only contains the service endpoint, we need to construct for instance, a "*GetCapabilities*" request from this URL (Lines 12–14 in Fig. 3), and then examine the new URL *u'* using the above pattern matching strategy to determine whether it belongs to our search target.

After the service type of the URL *u* is identified, its geolocation in terms of latitude and longitude is resolved through analyzing its IP address. This operation is to support map visualization in the GUI. The service quality of this service is also examined to support a better search experience from the end user. Then *u* is pushed to *l*, storing all the recently found URLs for real-time update of crawling status in the front end. *l*'s length is dynamic; it will not be cleared up until a client's request comes and harvests everything in *l*. Next, URLs on the webpage that *u* is referring to are extracted (Lines 15–19 in Fig. 3). The URLs includes not only the hyperlinks, but also those residing on the webpage as plain text. This web signature of service endpoints was identified by preliminary studies (Li et al., 2010). Finally all extracted URLs will be inserted into the crawling queue for further crawling (Line 19 in Fig. 3) and the URL *u* will be marked as crawled to avoid repeated visit by other crawler agents (Line 20 in Fig. 3).

This pattern-matching based crawling algorithm uses a hybrid searching strategy that combines depth-first and breadth-first crawling. As shown in the crawling expansion tree in Fig. 5, the crawling subtree at width *i* is processed ahead of those at crawling width *j*, where $i<j$, and a new set of seeds will not be added until all possible URLs linked by seeds inserted earlier have been visited. While within each subtree, the URLs are visited in the sequence of their depth, with a webpage closer to the seed URLs being visited sooner. This way, PolarHub is configured that URLs that are considered to be more probable links to a web service will be visited first. The crawling expansion tree can also easily be cut-off if we are searching for the wrong part of the Web with very few services found.

4.3. System workflow through asynchronous processing

4.3.1. Start and stop a task

Fig. 6 demonstrates the workflow for starting a web crawling task. Initially, a task *i* is configured by an authenticated user and the "start" request is sent to a controller responsible for starting the task (Step 1). This *StartTaskController* then obtains user information from the request session and checks whether a user is authorized or not (Steps 2 and 3). Rules are pre-set that only administrator can start unlimited number of tasks, other authenticated user is only allowed to start no more than *k* ($k = 5$) tasks. Once the user information is returned, the *StartTaskController* checks if the user has exhausted the upper limit of tasks they are allowed to start (Step 4). If not, a new task DAO is generated to save all attributes. A *TaskDAO* object (its class structure is shown in Fig. 2) needs to collectively define the rules for navigating the Web (Step 5). Specifically, attribute "depth" is the desired crawling depth to record the number of links away from the seed link the crawler wants to exploit. If this value is low, the crawling scale is constrained. When depth = 1, only webpage whose URLs are directly extracted from the seed webpage will be visited by the web crawler. When depth is set to a high value, more portions of the Web will be accessed. The possibility of finding more required dataset/services may be higher, but only if the crawler is searching at the part of the Web that has relevant information. Deep searching will certainly raise crawling expense. Overall, there should be a tradeoff between the scope of searching (determined by crawling depth and width) and the computing resources used. Experiments are conducted in Section 5 to demonstrate the effect of crawling depth and width to the number of dataset found.

After a *TaskDAO* object is created (Step 6), it is passed to the *CrawlerConfigurer* class to initiate a controller that manages multiple crawlers to accomplish crawling task *i* (Steps 7 and 8). Next the *CrawlControler* object starts crawlers in a non-blocking mode. This means that the start function does not need to wait until a crawler finishes crawling to return. Since crawling is usually a long process, the program will hang frequently and the GUI will freeze up if a blocking

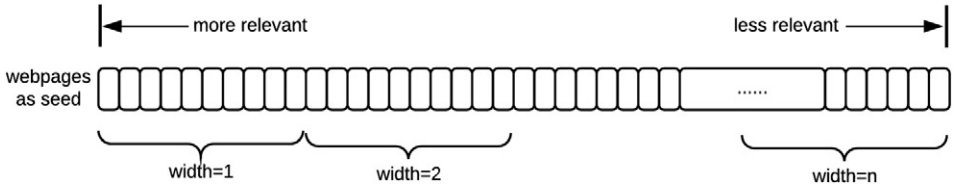


Fig. 4. Determination of crawling width from a list of seed URLs.

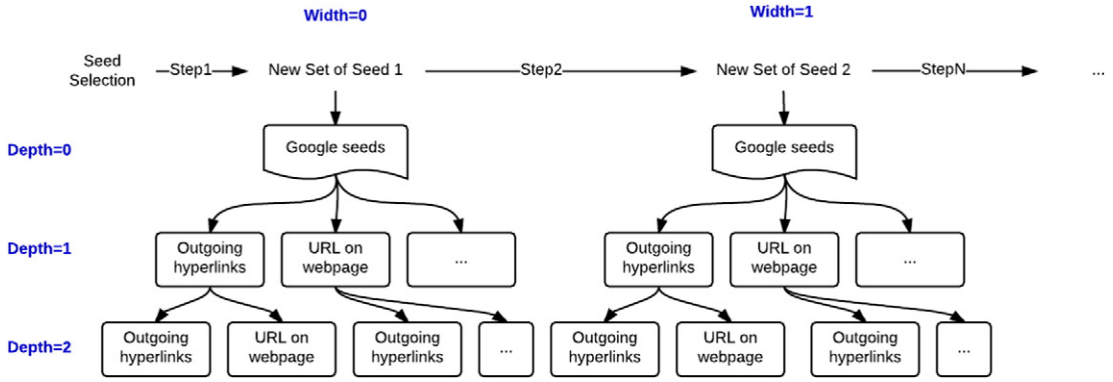


Fig. 5. Crawling expansion tree.

strategy is used. A non-blocking strategy can resolve this issue by cutting the tie between the processes in the front- and back- ends. The status of the crawler is monitored by the *CrawlController*, which is registered as a global application attribute in *ServletContext* and made available in the whole lifecycle of the web application (Step 10). Up to this point, a task has started, and crawler agents are assigned to work in parallel. The "*StartTaskController*" will then update the database to set the task status to be "active" (Step 11) and set the task starting time as well (Step 12).

In comparison to starting a task, stopping a task is straightforward. Once a stop command is sent from the client, the *CrawlController* changes its 'isAlive' property to *false*. The crawler agents working in the backend will detect this status change and shut down the crawling threads after cleaning up objects used for the crawler.

4.3.2. Real-time status monitor enabled by a hybrid client pull and server push strategy

A prominent feature of the crawling tool is real-time update of crawling status for any running task. Real-time means that the client receives update from the server instantly once new results are found. Such a feature is enabled by a combined use of asynchronous client-pull and server-push strategy. In most web applications, the client-server communication is normally accomplished using client-pull strategy. That is, a client checks the latest data from the remote web server at a user-defined interval. Ajax (Asynchronous JavaScript and XML) is widely adopted in this client-pull model because it allows incremental update of the GUI without the need to reload the whole page (Li, Yang, & Raskin, 2008), thereby offering a richer and more-responsive user experience. However, because Ajax does not know how frequently a server changes, it needs to constantly ping the server to acquire progressively available data, leading to a big waste of bandwidth and server resources. If the pulling frequency is set to low, there will be missing updates. To meet the challenges of Ajax, the server-push strategy emerges (Russell, 2006). Different from client pulling, once a request arrives, the server holds it until its status changes, i.e. requested data becomes available, to respond back to the client. This way, heavy traffic between client and server can be avoided. However, for a long running and constantly updating process like crawling, it is not sufficient to relying solely on

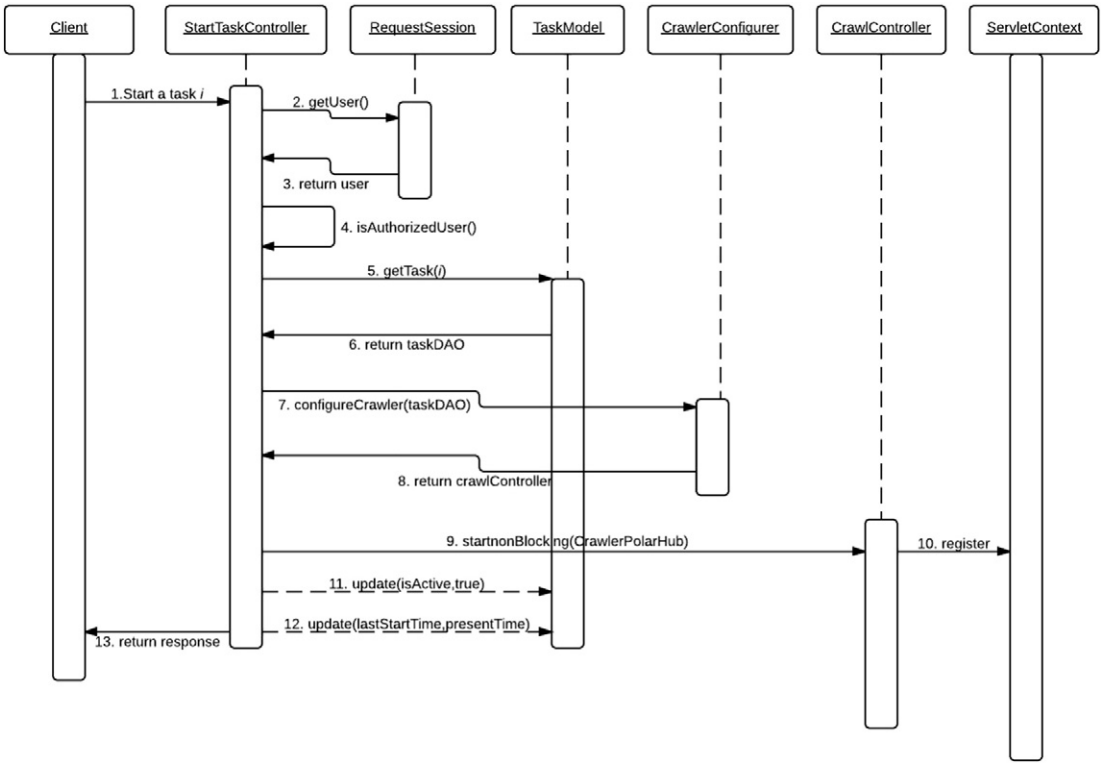


Fig. 6. Sequence diagram for starting a crawling task.

client pull or server push. Although client pull initiates requests and creates a connection channel for receiving data, requests that are too frequent impact server performance. Though a server-push approach alleviates client-server traffic, it is based on a passive response mode, which cannot keep pushing new data unless a new request comes. Therefore, a hybrid client pull and server push strategy, also known as asynchronous polling, is adopted. The client uses Ajax to send requests to the server, and the server will wait for an update, like if crawler finds some new target services. Once an update occurs, a complete response message with full data will be delivered to the client for visualization. If there is no new data in the given time, the server will notify the client that the request has expired. After receiving a response, the client immediately creates a new request and sends it to the server. This way, a low latency and efficient transmission model can be established.

Fig. 7 describes this procedure in detail. Initially, the *Asynchronous Communication Controller* receives a user query about the real-time crawling status of a task *i* (Step 1). This controller then retrieves the *CrawlController* of task *i* from the global attribute pool maintained by *ServletContext* (Step 2). Next, all properties about this task are read from the backend database and returned to *CrawlController* as a *TaskDAO* object (Steps 3 and 4). At this time, the communication controller checks if there are more data services found by the crawler agents (Steps 5 and 6). If so, these recent found data services would be returned as a list and then send back to the client (Steps 7–11). If not, the server alters to the asynchronous mode by initializing an asynchronous context (Steps 12 and 13), within which the request is held and a thread named *AsynchRequestProcessor* is executed (Step 14) to keep checking the backend for new datasets (Step 15). A message updating the status of the backend retrieval process will be sent back to the client after each step (Step 16). If a dataset arrives in the given time, the *CrawlController* will extract the crawling results from the *TaskDAO* object (as in Steps 3 and 4) and send them back to the client (Steps 18 and 19). Otherwise, the *CrawlControler* will prepare an empty response and notice the client (Step 17).

Following this workflow, the real-time update of crawling status and crawling results can be implemented very efficiently.

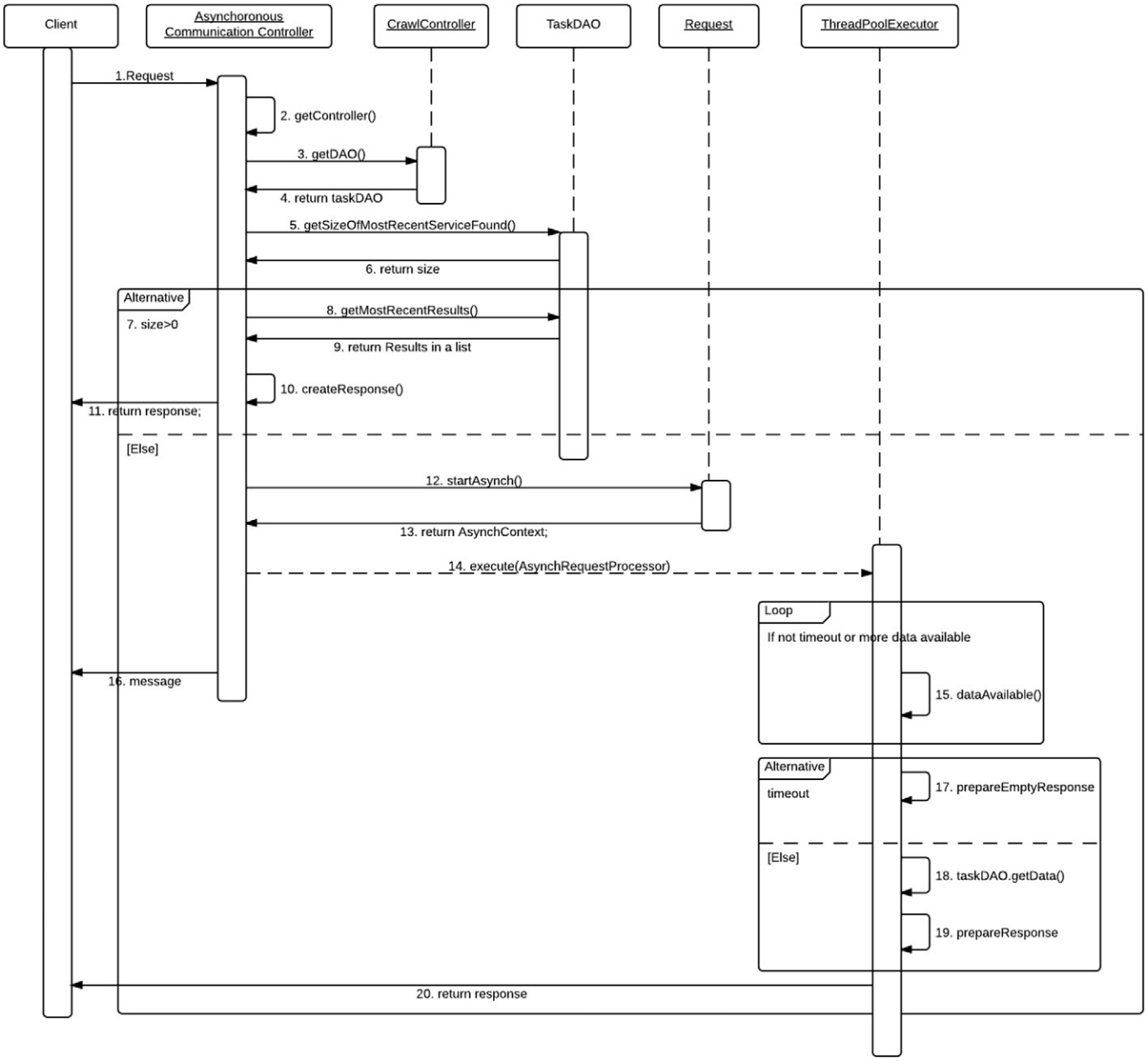


Fig. 7. Sequence diagram for real-time update of crawling results.

5. Graphic User Interface (GUI)

Fig. 8 demonstrates the GUI of the PolarHub system. The main dashboard contains three panels: task panel (box 1), service distribution map panel (box 2) and service panel (box 3). The task list displays all the crawling tasks a user is authorized to operate. There are three levels of security groups for end users. The guest user group is only able to view results of some sample tasks; they are not allowed to download the data services. The authorized user group will have full control of a task they created. System administrators can access and operate on any task created by any user and are also responsible for managing role of different user groups.

The web crawling cycle can be defined in the PolarHub. First, an authorized user creates a crawling task by providing search keywords, configuring crawling depth, width and other parameters, such as service type of interest. At present, PolarHub can discover eight types of geospatial web services: OGC WMS, WFS, WPS, CSW, WCS, SOS, WMTS and WCPS. Once created, the user can start the task on PolarHub and monitor the crawling status in real time. For example, as a new service is identified, a balloon will be popped up in the map component (box 2) to display the host location for this service. The service link will at the same time appear in the service table below the map (box 3). Each service will be assigned a quality score, ranging from 0 (low quality) to 100 (high quality) according to FGDC (Federal Geography Data Committee) service quality evaluation mechanism (box 4). The user can also view the URLs currently being crawled (box 5) by clicking "Realtime crawling" button in the task panel (box 1) and some statistical information by clicking "Realtime statistics" (box 6).

PolarHub provides other functions, including data search and miscellaneous crawling. The data search function provides a search interface to allow a user to search for interested services from PolarHub's service repository. If the service already exists, there is no need to re-crawl the Web. The miscellaneous crawling tool is to establish a direct connection between PolarHub to known data centers to harvest data service information through this customized crawling tunnel.

By providing these functions, PolarHub has successfully transformed traditional crawling paradigm from a daemon process running in the backend to an open, dynamic, and interactive process. The high interactivity lies in that a user can define according to their own interest the topic and type(s) of data to crawl, they can also freely choose the crawling depth and width according to the characteristics of their own task. During the crawling process, a user can monitor the crawling status, obtain statistical information, view the crawling results in real time, as well as stop a task once needed data is found to avoid waste of resources.

Next section discusses the impact of different crawling settings to the effectiveness of service discovery.

6. Results and analysis

In this section, we first analyze the effect of crawling depth and width to the crawling performance of PolarHub. Then, we introduce the spatial distribution pattern of all geospatial data services identified by PolarHub.

6.1. Impact of crawling depth and width to the search performance

Fig. 9 demonstrates the number of OGC WMSs found in two tasks that search using keyword "climate" (Fig. 9a) and "population" (Fig. 9b) at different width and depth settings. Very interesting patterns can be observed. First, as the crawling depth increases, more WMSs can be identified, regardless of the parameter settings for crawling width. This

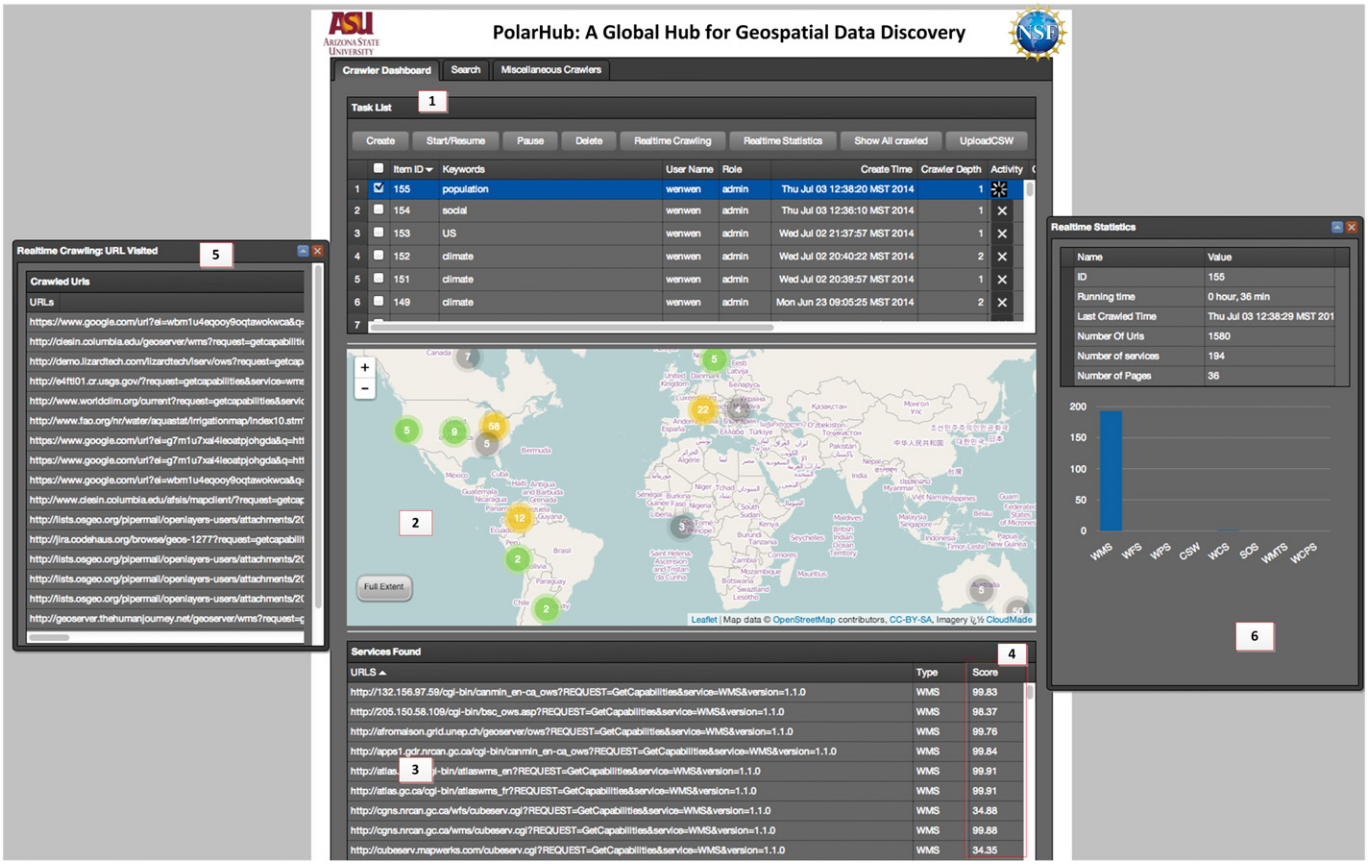
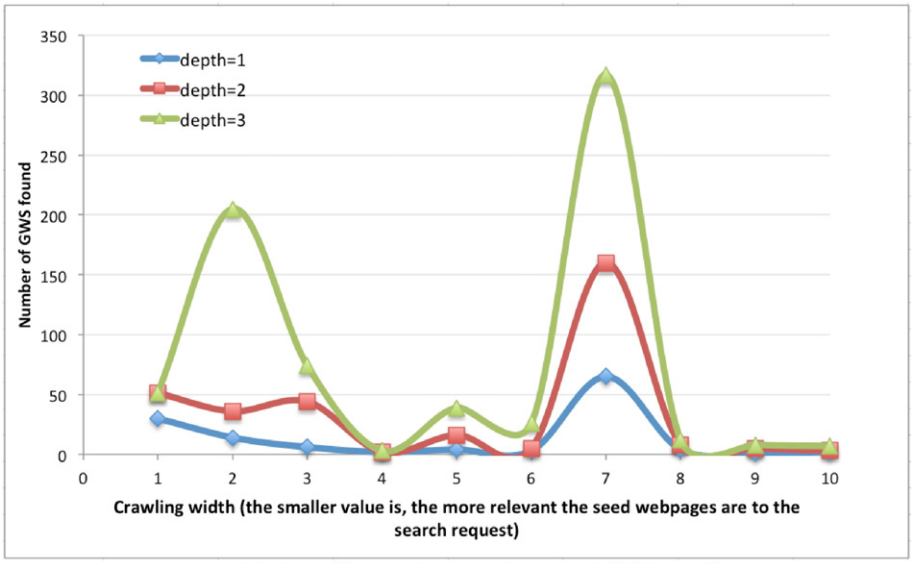
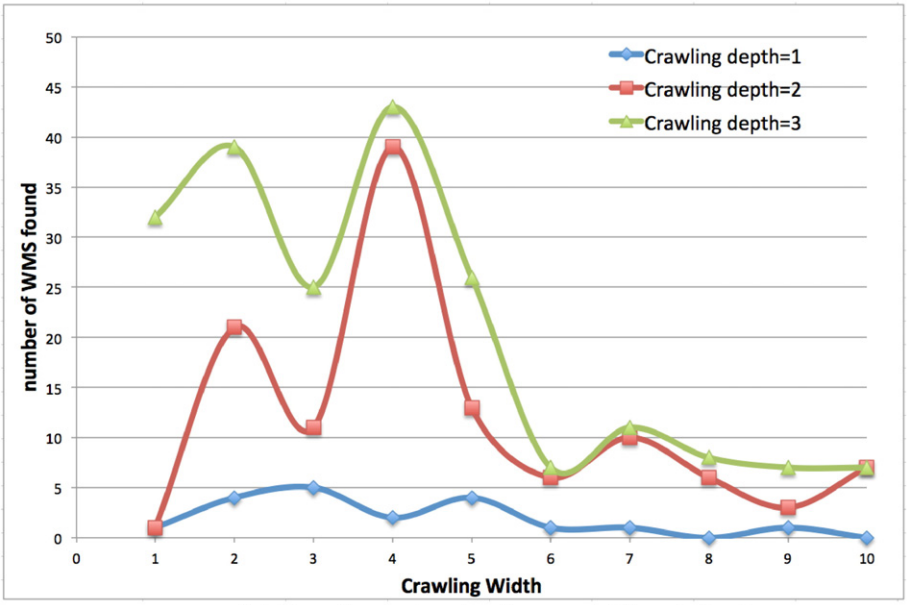


Fig. 8. A snapshot of PolarHub portal.
(http://polar.geodacenter.org/polarhub)



(a) Crawling task using keyword "Climate"



(b) Crawling task using keyword "Population"

Fig. 9. The impact of crawling depth and width to the number of services found. (a) Crawling task using keyword "Climate". (b) Crawling task using keyword "Population".

finding is confirmed in the figures by the fact that green lines (Crawling depth = 3) are always on top of red lines (Crawling depth = 2), and red lines sit always on top of the blue lines (Crawling depth = 1). This is not surprising: higher crawling depths visit the same set of sites as lower crawling depths in addition to some that are further away from the starting site. Therefore, we can conclude that the bigger the crawling depth is, more data services are likely to be found. However, crawling more webpages also means a substantial increase in system resource use. Taking the search using population data as an example, the total number of webpages (n_w) being crawled is 292, 11,679 and 138,621 when the crawling depth is 1, 2 and 3, respectively. The discovery rates, or the ratios between number of services found (n_d) and n_w, decrease dramatically from 6.5% for crawling at depth = 1 to 1% for depth = 2, and 0.15% for depth = 3. The same pattern applies to the crawling task with "climate" as the keyword. Discovery rates dropped from 5% to 0.3% from depth 1 to depth 3. Therefore, carefully select a crawling depth will avoid the crawler from being swamped in aimlessly crawling. In tests, a depth of 2 is most effective. A depth of 3 or higher is not viable because the large consumption of system resources.

However, the services identified with changing crawling width do not show similar patterns. Crawling width determines the seed webpages to start the crawling task. These seed webpages are organized by the relevancy of the webpage with the search keyword. The smaller the width is, the more relevant the seed webpages are ranked. From the result, we cannot see a pattern in whether a webpage that is considered to be more relevant to a search keyword actually contains more target data services. In Fig. 9(a), there is a burst in the number of services found when crawling depth is 3 and crawling width is 2. Similarly, even on the crawling seeds that ranked 7th in importance (crawling width = 7), most services were found from the group of seed webpages. This alerts the different search patterns that a crawler PolarHub should adopt from what a general user search on popular search engines. In a user manual search, a user may only review a few top most result records to determine if a search engine is effective or not (Jansen & Spink, 2003). In PolarHub search for geospatial data resources, however, a larger scope should be exploited in order to make effective data discovery. Fortunately, because PolarHub works under an automatic fashion, the search in a larger cyberspace could be much easier

accomplished. However, although a large scope search is necessary for PolarHub by setting a larger crawling width, there shall be an upper limit. From both experimental results demonstrated in Fig. 9 along with other experiments we conducted, it was observed that the majority data services (~90%) are found when the crawling width is less than eight. When the width is larger than 9, the crawling process will be much longer with very limited resources found. Therefore, a crawling width of eight is recommended to use in the crawling tasks.

6.2. Geographical distribution of geospatial data services found by PolarHub

Fig. 10 demonstrates the spatial distribution pattern of all the OGC services identified using PolarHub from over 100 user created tasks. This density map is generated using 150 sq miles as search radius, based on the point data in which each location refers to a web service and the attribute field "NumOfServices" provides the number of services clustered at that location. The color from blue to red shows the service density going from low to high. At present, PolarHub has collectively discovered over 40,000 OGC services with 1.5 million unique data layers that come from 82 countries. This number beats most, if not all, similar works in the geospatial field.

Fig. 10 shows clear patterns for the clustered distribution of OGC web services. One hot spot region is in Europe, which contains several hotspots (in red) connecting with each other. Though European countries are smaller in extent, but they are very active in promoting the spatial interoperability among heterogeneous geospatial data. This substantial progress is probably due to the big push from the European Union (EU) to develop the EU-wide spatial data infrastructure – INSPIRE (Craglia & Annoni, 2007) to facilitate the public access of spatial information. Other hotspots can also be identified from the map. In the US, there exist six major hotspots, from east to west, they are government agencies, including NASA, NOAA, USGS etc. in DC area; the Illinois State Geological Survey in Champaign, Illinois; National Center for Atmospheric Research in Boulder, CO; Earth Data Analysis Center in the University of New Mexico; the Arizona State Geological Survey in Phoenix, AZ; and NOAA's west coast regional node in Pacific Grove, CA. In addition to these services distributed in Europe and in the US, there is another hotspot located in central Australia.

Fig. 11 shows more statistics and lists top 10 countries in terms of number and types of data services they are providing. United States is the top service provider country, followed by Germany. These two countries in total host over 50% of the total number of services found worldwide. Then the order is France, Spain, Netherlands, Belgium, United Kingdom, Italy, Czech Republic and Norway. 90% of these countries are in Europe; this result echoes results from hotspot analysis. Fig. 11 also categorizes the data according to their service types. Overall, the majority of the data services are WMS, then WFS. There are only a small number of WPS, WCS, CSW and WMTS located. So far, PolarHub has not identified any WCPS from about 100 user tasks. The distribution that WMS and WFS comprise the major portion of the bar is similar across all the countries analyzed in the Figure. In this analysis, we can tell that WMS is the most popular open standard used in sharing the earth science data, due to its easy implementation and the widespread availability of earth science data in raster format. WFS's number also substantially increases over the past years and has been supported for visualization in different cyberinfrastructure portals, such as the NSF polar cyberinfrastructure portal (Li et al., 2014).

7. Conclusion and discussion

Big data brings us both opportunities and challenges (Katina & Miller, 2013). To increase the visibility and usability of hidden geospatial data in the cyberspace to better facilitate scientific applications, we developed a viable solution, the PolarHub, to support the automatic discovery of distributed geospatial resources. Innovative web mining strategies, including the meta-crawling algorithm, method to restrain the search scope and the pattern-based information extraction, were introduced to enable a high performance, scalable, sustainable, collaborative, and interactive platform for active discovery of OGC web services. In addition, PolarHub is established on a pure service-oriented architecture (SOA) in which Data Access Object (DAO) design patterns are extensively used. This design pattern separates persistent data storage and business logic, and therefore makes the PolarHub system highly extensible to support data discovery in other formats or in other domains. The advanced asynchronous communication strategy, which combines client-pull and server-push, ensures high efficiency of the crawling system. As we are entering the era of big (geo) data, PolarHub is becoming a new "surfboard" for this data deluge to support insightful geospatial data acquisition and analytics.

At the present, PolarHub is up and running and is benefiting various science communities that demands geospatial data. A performance evaluation on how well PolarHub has enabled the data accessibility

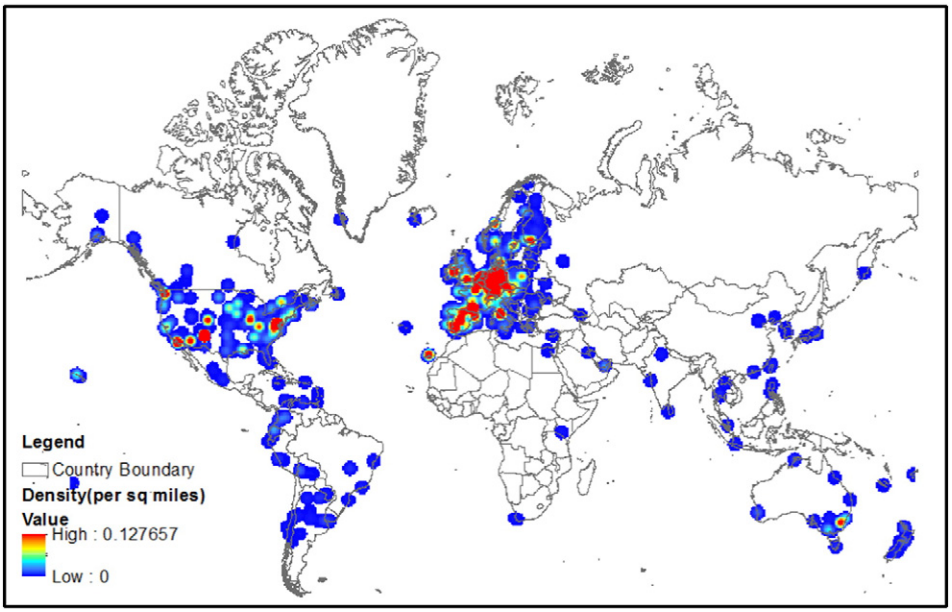


Fig. 10. Density map demonstrating the spatial distribution pattern of PolarHub-found OGC services. The unit of density is square mile.

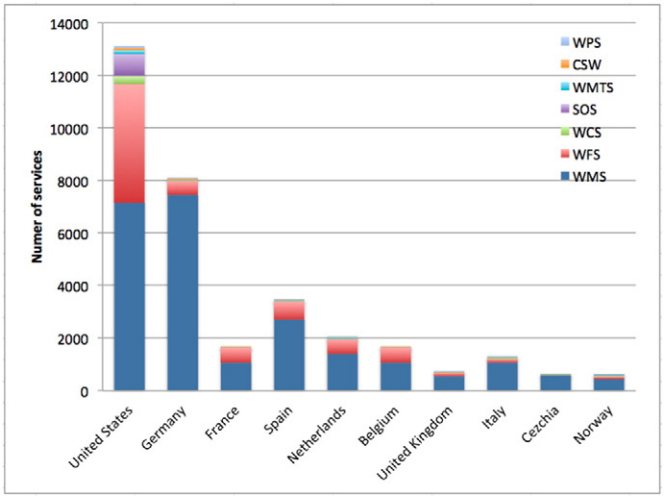


Fig. 11. Statistics of data services found by PolarHub.

was conducted recently. We compared the number of services hosted in PolarHub and other major SDI portals, such as Data.gov, ESRI geoportal, Global Earth Observation System of Systems (GEOSS), and the Spatineo catalog. It was found that PolarHub hosts the most data services and this number is 50% more than the second most powerful tool, Spatineo, and is much higher than that hosted in Data.gov, ESRI geoportal and GEOSS clearinghouse. We expect that PolarHub will become a new form of spatial data infrastructure (SDI) which integrates the large-scale web crawling to provide geospatial data services not only locally, but also regionally and globally.

We are also working on enhancing the PolarHub tool in terms of its crawling algorithm, the crawling framework and crawling user interface. Specifically, we are extending the crawling algorithm to support discovery of popular scientific data, such as NetCDF or HDF. We will develop a more intelligent crawling engine through the integration with geospatial domain knowledge, e.g. utilizing Semantic Web and ontology techniques (Li et al., 2008; Li et al., 2011; Raskin, Zhou, & Li, 2010). Based on the services found by PolarHub, we will develop advanced geospatial orchestration services which will combine and enable a chain of workflow to support complex scientific analysis. We will also enhance PolarHub from crawling English-dominant webpages to that of other languages, such as Germany and Spanish, to further enlarge its search scope. Finally, PolarHub will be customized to support efficient data discovery in domain-specific science areas, such as polar (Li, Bhatia, & Cao, 2015) or atmospheric science (Li, Shao, Wang, Zhou, & Wu, 2016).

Acknowledgements

This project is supported by the National Science Foundation (PLR-1349259; BCS-1455349; and PLR-1504432), and the Open Geospatial Consortium (OGC-FP00006698).

References

de Andrade, F. G., de Souza Baptista, C., & Davis, C. A., Jr. (2014). Improving geographic information retrieval in spatial data infrastructures. *GeoInformatica*, 1–26.

ANSI (1995). Information retrieval application service definition and protocol specification for open systems interconnection. *ANSI/NISO Z39.50–1995 (ISO 23950)*. American National Standards Institute (ANSI) (available at http://lcweb.loc.gov/z3950/agency/document.html).

Asrar, G., & Dozier, J. (1994). EOS. *Science strategy for the earth observing system. 1*, . AIP Press (128 pp.).

Bernard, L., Kanellopoulos, I., Annoni, A., & Smits, P. (2005). The European geoportal–-One step towards the establishment of a European spatial data infrastructure. *Computers, Environment and Urban Systems*, *29*(1), 15–31.

Busby, J. W. (2008). Who cares about the weather?: Climate change and US national security. *Security Studies*, *17*(3), 468–504.

Cho, J., & Garcia-Molina, H. (2002, May). Parallel crawlers. *Proceedings of the 11th international conference on World Wide Web* (pp. 124–135). ACM.

Craglia, M., & Annoni, A. (2007). INSPIRE: An innovative approach to the development of spatial data infrastructures in Europe. *Research and theory in advancing spatial data infrastructure concepts* (pp. 93–105).

DataONE (2014). https://www.dataone.org/news/nsf-awards-15-million-dataone

Di, L., Chen, A., Yang, W., & Zhao, P. (2003). *The integration of grid technology with OGC web services (OWS) in NWGISS for NASA EOS data.* GGF8 & HPDC12, 24–27.

Farmer, C. J., & Pozdnoukhov, A. (2012). Building streaming GIScience from context, theory, and intelligence. *Proceedings of the workshop on GIScience in the Big Data Age* (Columbus, Ohio, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.298.3723&rep=rep1&type=pdf. (Last accessed on July 16, 2016)).

FGDC (1998). Content standard for digital geospatial metadata, version 2.0. *Document FGDC-STD-001-1998, Federal Geographic Data Committee (FGDC)*. Metadata Ad Hoc Working Group.

Foster, I. (2005). Service-oriented science. *Science*, *308*(5723), 814–817.

Granell, C., Díaz, L., & Gould, M. (2010). Service-oriented applications for environmental models: Reusable geospatial services. *Environmental Modelling & Software*, *25*(2), 182–198.

Hart, J. K., & Martinez, K. (2006). Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, *78*(3), 177–191.

Heidorn, P. B. (2008). Shedding light on the dark data in the long tail of science. *Library Trends*, *57*(2), 280–299.

ISO (2003). Geographic information—Metadata. *ISO 19115:2003*. International Organization for Standardization (ISO).

Janowicz, K., Scheider, S., Pehle, T., & Hart, G. (2012). Geospatial semantics and linked spatiotemporal data – Past, present, and future. *Semantic Web*, *3*(4), 321–332.

Jansen, B. J., & Spink, A. (2003, June). An analysis of web documents retrieved and viewed. *International conference on internet computing* (pp. 65–69).

Kääb, A., Wessels, R., Haeberli, W., Huggel, C., Kargel, J. S., & Khalsa, S. J. S. (2003). Rapid ASTER imaging facilitates timely assessment of glacier hazards and disasters. *Eos, Transactions American Geophysical Union*, *84*(13), 117–121.

Katina, M., & Miller, K. W. (2013). Big data: New opportunities and new challenges [guest editors' introduction]. *Computer*, *46*(6), 22–24.

Kawanishi, T., Sezai, T., Ito, Y., Imaoka, K., Takeshima, T., Ishido, Y., ... Spencer, R. W. (2003). The Advanced Microwave Scanning Radiometer for the Earth Observing System (AMSR-E), NASDA's contribution to the EOS for global energy and water cycle studies. *IEEE Transactions on Geoscience and Remote Sensing*, *41*(2), 184–194.

Khalsa, S. J. S., Nativi, S., & Geller, G. N. (2009). The GEOSS interoperability process pilot project (IP3). *IEEE Transactions on Geoscience and Remote Sensing*, *47*(1), 80–91.

Klamma, R., Cao, Y., & Spaniol, M. (2008). Smart social software for mobile cross-media communities. *Multimedia semantics—The role of metadata* (pp. 87–106). Berlin Heidelberg: Springer.

Kottman, C. (1999). The OpenGIS abstract specification. *Topic13: Catalog services (version 4). OpenGIS project document 99–113*. OpenGIS Consortium Inc.

Lake, R., & Farley, J. (2007). Infrastructure for the geospatial web. *The geospatial web* (pp. 15–26). London: Springer.

Lakhani, K. R., Austin, R. D., & Yi, Y. (2010). *Data. Gov*. Cambridge, MA: Harvard Business School.

Langville, A. N., & Meyer, C. D. (2011). *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press.

LeVan, R. (2013). OpenSearch and SRU: A continuum of searching. *Information Technology and Libraries*, *25*(3), 151–153.

Li, W., Bhatia, V., & Cao, K. (2015). Intelligent polar cyberinfrastructure: Enabling semantic search in geospatial metadata catalogue to support polar data discovery. *Earth Science Informatics*, *8*(1), 111–123.

Li, W., Goodchild, M. F., & Raskin, R. (2014). Towards geospatial semantic search: Exploiting latent semantic relations in geospatial data. *International Journal of Digital Earth*, *7*(1), 17–37.

Li, W., Li, L., Goodchild, M. F., & Anselin, L. (2013). A geospatial cyberinfrastructure for urban economic analysis and spatial decision-making. *ISPRS International Journal of Geo-Information*, *2*(2), 413–431.

Li, W., Shao, H., Wang, S., Zhou, X., & Wu, S. (2016). A2CI: A cloud-based, service-oriented geospatial cyberinfrastructure to support atmospheric research. *Cloud computing in ocean and atmospheric sciences* (pp. 137).

Li, W., Yang, C., Nebert, D., Raskin, R., Houser, P., Wu, H., & Li, Z. (2011). Semantic-based web service discovery and chaining for building an Arctic spatial data infrastructure. *Computers & Geosciences*, *37*(11), 1752–1762.

Li, W., Yang, C., & Raskin, R. (2008). A semantic enhanced search for spatial web portals. *AAAI spring symposium: Semantic scientific knowledge integration* (pp. 47–50).

Li, W., Yang, C., & Yang, C. (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.

Liu, K., Yang, C., Li, W., Li, Z., Wu, H., Rezgui, A., & Xia, J. (2011, June). The GEOSS clearinghouse high performance search engine. *Geoinformatics, 2011 19th international conference on* (pp. 1–4). IEEE.

Lopez-Pellicer, F. J., Rentería-Agualimpia, W., Béjar, R., Valiño, J., Zarazaga-Soria, F. J., & Muro-Medrano, P. R. (2011). *Implantation of OGC geoprocessing services for geoscience.*

Malik, T., & Foster, I. (2012, July). Addressing data access needs of the long-tail distribution of geoscientists. *Geoscience and remote sensing symposium (IGARSS), 2012 IEEE international* (pp. 5348–5351). IEEE.

Manney, G. L., Santee, M. L., Froidevaux, L., Hoppel, K., Livesey, N. J., & Waters, J. W. (2006). EOS MLS observations of ozone loss in the 2004–2005 Arctic winter. *Geophysical Research Letters*, *33*(4).

Meier, W. N., Marquis, M., Kaminski, M., & Weaver, R. (2004). NASA EOS sensors demonstrate potential for multiparameter studies of Arctic Sea ice. *Eos, Transactions American Geophysical Union*, *85*(46), 481–489.

Nativi, S., Craglia, M., & Pearlman, J. (2013). Earth science infrastructures interoperability: The brokering approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *6*(3), 1118–1129.

Nebert, D. (2008, February). Global earth observing system-of-systems (GEOSS): A global SDI framework. *10th Int. Conf. Spatial data infrastructure (GSDI-10), St. Augustine, Trinidad*.

Nebert, D., & Whiteside, A. (Eds.). (2004). *OpenGIS Catalog Service Specification 2.0. OGC Document 04-021r2*.

Raskin, R., Zhou, N., & Li, W. (2010). Geoinformation knowledge representation and applications. *Advanced Geoinformation Science*, 276–299.

Rautenbach, V., Coetzee, S., & Iwaniak, A. (2013). Orchestrating OGC web services to produce thematic maps in a spatial information infrastructure. *Computers, Environment and Urban Systems*, *37*, 107–120.

Russell, A. (2006). Comet: Low latency data for browsers. https://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/ (Last accessed July 15, 2016)

Sample, J. T. (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*. (pp. 1–3).

Steffen, W., Jager, J., Carson, D. J., & Bradshaw, C. (2002). *Challenges of a changing Earth*. Springer.

Sui, D., & Goodchild, M. (2011). The convergence of GIS and social media: Challenges for GIScience. *International Journal of Geographical Information Science*, *25*(11), 1737–1748.

Ustin, S. L., Wessman, C. A., Curtiss, B., Kasischke, E., Way, J., & Vanderbilt, V. C. (1991). *Opportunities for using the EOS imaging spectrometers and synthetic aperture radar in ecological models.*

Van de Sompel, H., & Lagoze, C. (2001). The open archives initiative protocol for metadata harvesting – Version 1.0. www.openarchives.org/OAI/1.0/openarchivesprotocol.htm

Wang, S. (2010). A cyberGIS framework for the synthesis of cyberinfrastructure, GIS, and spatial analysis. *Annals of the Association of American Geographers*, *100*(3), 535–557.

Weiser, A., & Zipf, A. (2007). Web service orchestration of OGC web services for disaster management. *Geomatics solutions for disaster management* (pp. 239–254). Berlin Heidelberg: Springer.

Williams, D. N., Drach, R., Ananthakrishnan, R., Foster, I. T., Fraser, D., Siebenlist, F., ... Sim, A. (2009). The earth system grid: Enabling access to multimodel climate simulation data. *Bulletin of the American Meteorological Society*, *90*(2), 195–205.

Yang, C., Raskin, R., Goodchild, M., & Gahegan, M. (2010). Geospatial cyberinfrastructure: Past, present and future. *Computers, Environment and Urban Systems*, *34*(4), 264–277.