Egyptian Informatics Journal (2013) **14**, 27–36

Cairo University

**Egyptian Informatics Journal**

www.elsevier.com/locate/eij
www.sciencedirect.com

**ORIGINAL ARTICLE**

# Maintaining the search engine freshness using mobile agent

**Marwa Badawi [a,*], Ammar Mohamed [a], Ahmed Hussein [b], Mervat Gheith [a]**

[a] *Institute of Statistical Studies and Research, Cairo University, Cairo, Egypt*
[b] *Faculty of Computers and Information, Cairo University, Cairo, Egypt*

Received 23 June 2012; revised 12 November 2012; accepted 20 November 2012
Available online 21 December 2012

**KEYWORDS**

Search engine;
Mobile crawling;
Distributed indexing;
Web page change detection

**Abstract** Search engines must keep an up-to-date image to all Web pages and other web resources hosted in web servers in their index and data repositories, to provide better and accurate results to its clients. The crawlers of these search engines have to retrieve the pages continuously to keep the index up-to-date. It is reported in the literature that 40% of the current Internet traffic and bandwidth consumption is due to these crawlers. So we are interested in detecting the significant changes in web pages which reflect effectively in search engine's index and minimize the network load. In this paper, we suggest a document index based change detection technique and distributed indexing using mobile agents. The experimental results have shown that the proposed system can considerably reduce the network traffic and the computational load on the search engine side and keep its index up-to-date with significant changes.



## 1. Introduction

Indexing the Web has become a challenge due to the web's growing and dynamic nature. Currently it is estimated that web contains approximately 50 billion publicly accessible/indexable web documents distributed all over the world on thousands of web servers, while the deep web (dynamically generated documents, intranet pages, web-connected databases, etc.) is almost three orders of magnitude larger [1]. According to [2], the web is very dynamic and 40% of its contents change daily. Web crawlers are used to recursively traverse and download web pages for search engines to create and maintain the web indices. According to [3], the need of maintaining the up-to-date pages in the indices causes a crawler to revisit the websites recursively. Hence, the resources like CPU cycles, disk space, and network bandwidth, etc., become overloaded and sometimes a web site may crash due to such overload on these resources. One study [4] reports that about 40% of current internet traffic and bandwidth consumption is due to the web crawlers. The current cooperation schemes between the search engine and web server can be divided in two main groups as depicted in [5]: polling schemes and interrupt (or push) schemes. A crawler may use one of them or a combination of different schemes. In the polling (or pull) schemes, the search engine periodically requests data from the web server, based on search engine policies. In the interrupt

* Corresponding author. Tel.: +20 1009620075.
E-mail address: marwa_badawi_claes@yahoo.com (M. Badawi).
Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier


http://dx.doi.org/10.1016/j.eij.2012.11.001

(or push) schemes, the web server begins a transaction with the search engine whenever it is necessary. This is similar to the relationship between the main processor and a hardware device (network card, scanner, etc.) in a modern computer. However, with the great expansion of the web, the currently centralized crawling and indexing approach based on pull schemes appears inadequate. Since crawlers are no longer able to download web pages with the daily rate required to maintain an updated index of the web. A study suggests that no search engine succeeds coverage of more than 16% of the estimated web size [6]. More specifically; the centralized crawling and indexing approach has a shortage due to the following reasons [7]:

- *Centralized data access:* The task of crawling data is highly centralized. It uses HTTP request and reply paradigm for every page downloaded to create and maintain the search engine's index. Each of these requests and replies requires a separate TCP connection. Given that the web today has approximately 50 billion publicly pages, the latency involved in establishing these connections quickly adds up. Further, the estimated amount of data in all the web pages is of the order of tens of terabytes and continues to grow. Since a search engine has to frequently re-crawl web pages to account for any changes, the required network bandwidth is tremendous.
- *Centralized page filtering:* Crawlers download the entire contents of a web page, including useless information such as scripting code, HTML tags and comments, which are unnecessary for the document indexing.
- *Centralized indexing:* The indexing strategy is centralized too where all the downloaded pages are processed locally at the search engine to generate inverted indices, which require a lot of storage and processing power.
- *Centralized change detection:* The crawler detects changes between the last version of downloaded web pages and the old one locally at the search engine side.
- *Uncompressed data:* Documents are usually downloaded by the crawlers uncompressed, increasing in this way the network bandwidth.
- *Asynchronous updating:* The vast size of the web makes it impossible for crawlers to keep up with document changes. The revisiting frequency for non-sponsored documents, in some of the biggest commercial search engines, varies from 4 to 6 weeks [8]. As a result, search engines provide the clients old content. The ideal case would to synchronize the update of the search engine's index with the web page's actual change frequency.
- *Insignificant change detection:* Changes occurring in web pages are best classified as content change (e.g., deletions, additions and modification of text), layout or structure change (e.g., changes in the position of elements in the page), and attributes change (e.g., changes in fonts and colors). The significant changes that actually reflected in the search engine's index are the content changes. But the crawler in this centralized approach is nave where it transmits over the network both significant changes (content changes) and insignificant changes (structure and attributes changes), therefore it cause network overload and useless indexing for insignificant changes.

In this paper, we propose a document index based web page change detection technique and distributed indexing strategy by utilizing the mobile agent technology. The proposed distributed web crawling and indexing system (DWCIS) handles the above mentioned shortages in centralized approach by minimizing network utilization, to keep up the search engine's index up-to-date with document significant changes in real time by performing on-site monitoring and to minimize the computational load on the search engine side.

The rest of the paper is organized as follows. Section 2 provides an overview of centralized or traditional search engine design. The related work is discussed in Section 3. Section 4 describes the proposed architecture of the distributed web crawling and indexing system (DWCIS) and its work flow. Section 5 describes the experimental setup and Section 6 shows the experimental results and discussion. Finally, we conclude in Section 7.

## 2. An overview of traditional search engine design

A search engine usually contains spiders, a web page repository, an indexer, search indexes, a query engine, and a user interface. These components are described in the following [9].

- *Spiders*: also referred to as web robots, or crawlers, are the programs behind a search engine that retrieve web pages by recursively following URL links (Uniform Resource Locator) in pages using standard Hyper Text Transfer Protocol (HTTP). First, the spiders read from a list of starting-seed URLs and download the documents at these URLs. Each downloaded page is processed, and the URLs contained within it are extracted and added to the queue. Each spider then selects the next URL from the queue and continues the process until a satisfactory number of documents is downloaded or local computing resources are exhausted. To improve speed, spiders usually connect simultaneously to multiple web servers in order to download documents in parallel, either using multiple threads of execution or asynchronous input/output.
- *Web page repository:* Documents retrieved by the spiders are stored in a repository of web pages. To reduce the needed storage space, the pages are often compressed before being stored. The repository is usually in the form of a database, but it is also common for small-scale search engines to simply store the documents as files.
- *Indexers*: An indexer processes the pages in the repository and builds an underlying index of the search engine. The indexer tokenizes each page into words and records the occurrence of each word in the page. The indexer is also used to calculate scores such as the term and document frequencies of each word, which can be used for search result rankings.
- *Inverted index:* The results from the indexer are then converted into an ''inverted index''. While the original indexing results map a document to a list of words contained within it, an inverted index maps a word to a list of documents containing the word. This allows fast retrieval of documents when a search query is passed to the search engine. The resulting searchable indexes are usually stored in a database.

- *Query engines:* A query engine accepts search queries from users and performs searches on the indexes. After retrieving search results from the indexes, the query engine is also responsible for ranking the search results according to content analysis and link analysis scores. It is also responsible for generating a summary for each search result, often based on the web page repository. The query engine in some search engines is also responsible for caching the results of popular search queries. After all the processing, the query engine generates and renders a search result HTML page and sends it to the user interface.

As shown in Fig. 1, the processes of the traditional search engine are totally centralized at the search engine side. Starting from the crawling process, change detection process to indexing process. Given that the web today has approximately 50 billion publicly indexable pages as mentioned before, this is huge load on the search engine side and the network bandwidth required is tremendous.

## 3. Related work

These authors [7,10,11] have initiated the concept of mobile crawling by processing the use of mobile agents as the crawling units. The proposed concept surpasses the centralized architecture of the current web crawling systems by distributing the data retrieval process across the network. Mobile crawlers are able to perform remote operations such as data analysis and data compression at the data source before the data is transmitted over the network. However their systems ignore the distributed indexing, index updating and web page change detection.

Brandman et al. [12] have studied the idea of how to make web servers more crawler friendly through that web servers export meta-data archives describing their content, so that crawlers can efficiently create and maintain large, ''fresh'' repositories. This meta-data includes the last modified date and size for each available file. This approach reduces the network bandwidth by sending only the modified pages after last crawling date, but in the same time it provides the search engine's index with significant changes (content changes) and insignificant changes (structure and attributes changes) together. These insignificant changes results in network overload and wasted resources for the search engine through re-crawling and re-indexing the web pages for insignificant changes. The authors here concern only with the search engine's index freshness and ignore the distributed crawling and indexing.

Yadav et al. [13,14] have proposed checksum (hash value) based content level change detection. At the time of page crawling, only comparison will be made to the text code of that page. The main drawback of this technique is that if any change in that value is detected for the actual copy on the web as compared to the local copy, regardless it is significant or not, the page will be refreshed or re-crawled. Hence this technique results in network overload and wasted resources for the search engine.

Artail and Abi-Aad [15] have proposed a web page change detection approach based on restricting the similarity computations between two versions of a given web page to the nodes with the same HTML tag type. Before performing the similarity computations, the HTML web page is transformed into an XML-like structure in which a node corresponds to an open-closed HTML tag. This tree structure uses a lot of storage space as well as causes a lot of inconvenience at time of refresh, as the tree structure has to be compared. Also this approach works only on the page types that can be transformed into an XML-like structure such as HTML pages.

Bal et al. [16,17] have proposed a novel indexing system based on mobile agents, which can filter out the HTML pages that have not been modified since last crawl through two web page change detection methods. The first method is the comparison of page sizes of web pages at the time of page change detection. The second one uses the last modification date of web pages. These methods have the same drawback like the hashing method in above discussed related work as any insignificant change will change both the page size as well as its last modification date. This leads to overloading the search engine with processing web pages that will not change the index.

These authors [18–20] have proposed a distributed crawling system based on mobile agents, but they do not take into

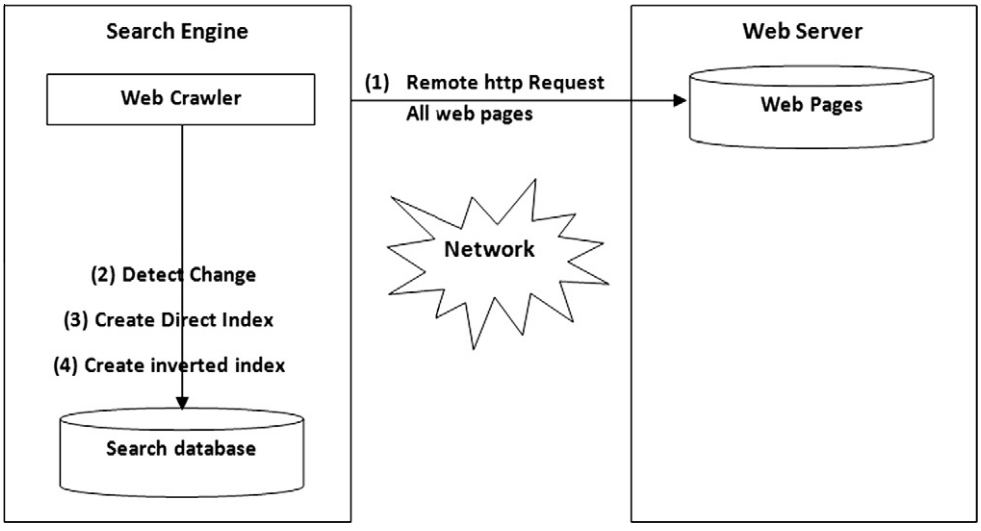


**Figure 1** Work flow of the traditional crawling system.

consideration either distributed indexing or web page change detection. In [21] a technique is proposed to utilize the users' browsing behavior at the crawling and indexing process so as to direct the crawler to download the important pages, which were not previously crawled. As the work attempts to index most of important pages based on user feedback.

These authors [22–29] have studied the concept of parallel crawling; multiple processes run in parallel to download pages. So that download rate is maximized, network load dispersion, network load reduction and scalability, but they did not consider distributed indexing and change detection.

## 4. The proposed distributed system architecture

We introduce a distributed web crawling and indexing system (DWCIS). The key idea of our architecture is based on the Master-Slave agent design pattern [30,31]. On the Master-Slave design pattern, a master agent delegates a task to be done on a given agency to a slave agent, in order to continue executing other tasks that cannot be interrupted. The slave agent visits the indicated agency, where it accomplishes the task and then returns to the source agency with the results. The master agent receives the results from the slave agent. Then, the slave one destroys itself. Fig. 2 shows the idea of our system as follows: instead of downloading the pages at a web server across the network; the search engine uploads an agent, called the mobile crawler to the web server. The mobile crawler processes the pages at the web server locally and sends back the results in a custom format to the search engine.

The main contribution of the proposed system is the creation of document index of web pages at the web server side. This proposed approach has three advantages: First, it is used as a change detection technique where it is more robust to non-significant web page changes than other existing change detection techniques that depend on page size, last modification date and hash value. Second, the document index of changed pages is returned to the search engine instead of the pages themselves. Therefore, it reduces the network load results from crawling. Third, it reduces the computational load at the search engine side because the document indices of the web sites are already created at the web server and the search engine has to only create the inverted indices.

The major components of our DWCIS as shown in Figs. 2 and 3 are:

- *Master Agent (MA):* The MA resides at the search engine side and performs various tasks. The major tasks of MA are slave agents (mobile crawlers) creation; delegation of URL's to the SA for crawling, dispatching SA to the web server, receiving the compressed document index from the slave agent, decompression of document index and provide search engine's index with document index.
- *Slave Agent (SA):* It is created at the search engine side then dispatched to reside at the web server side. It processes the assigned URL locally as follows: requests the web pages from the web server, generates the document index of the web pages, detect significantly changed web pages, compresses the generated document index and finally moves back to the search engine side carrying the compressed document index of the significantly changed web pages and sends them to the MA in an ACL message.

### 4.1. Work flow of DWCIS

We assumed that any web server wants to participate to our system should first register at the Master Agent through our administration site (Fig. 4). Then the Master agent has a

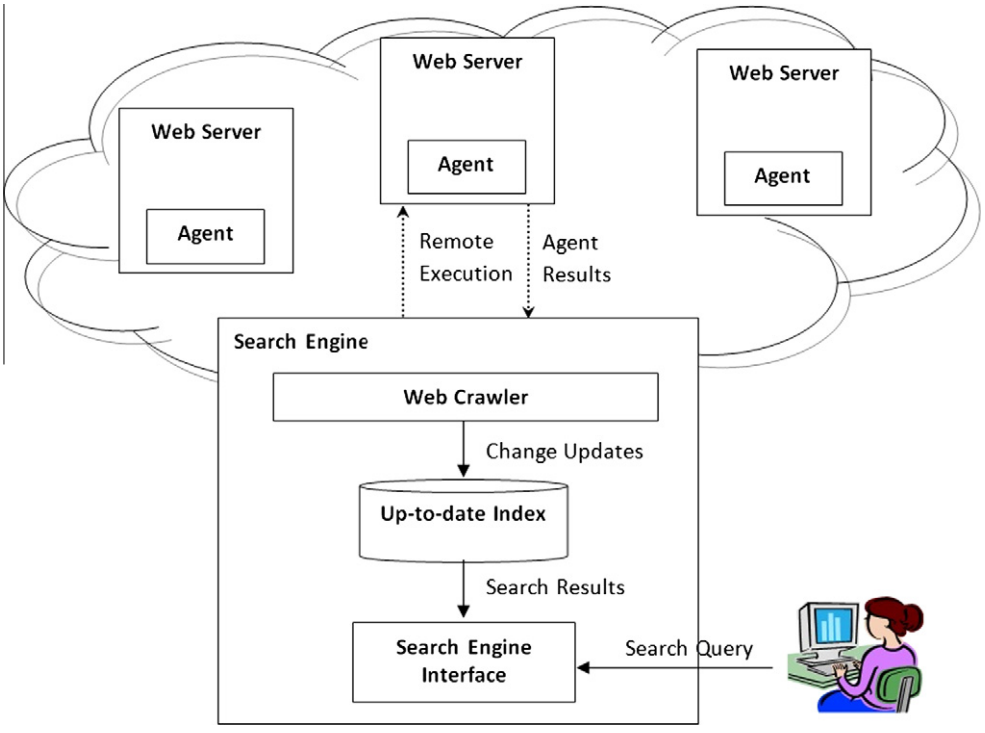


**Figure 2** Architecture of the proposed system.

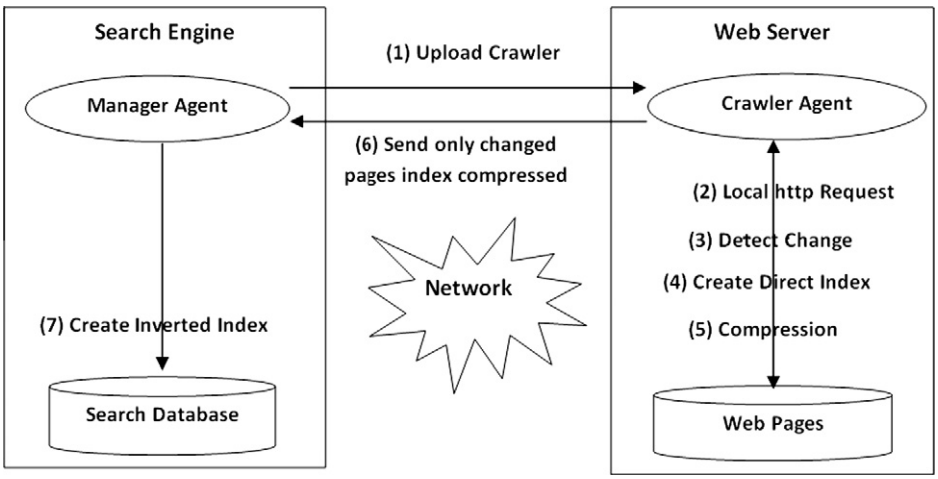


**Figure 3** Work flow of the proposed distributed system.

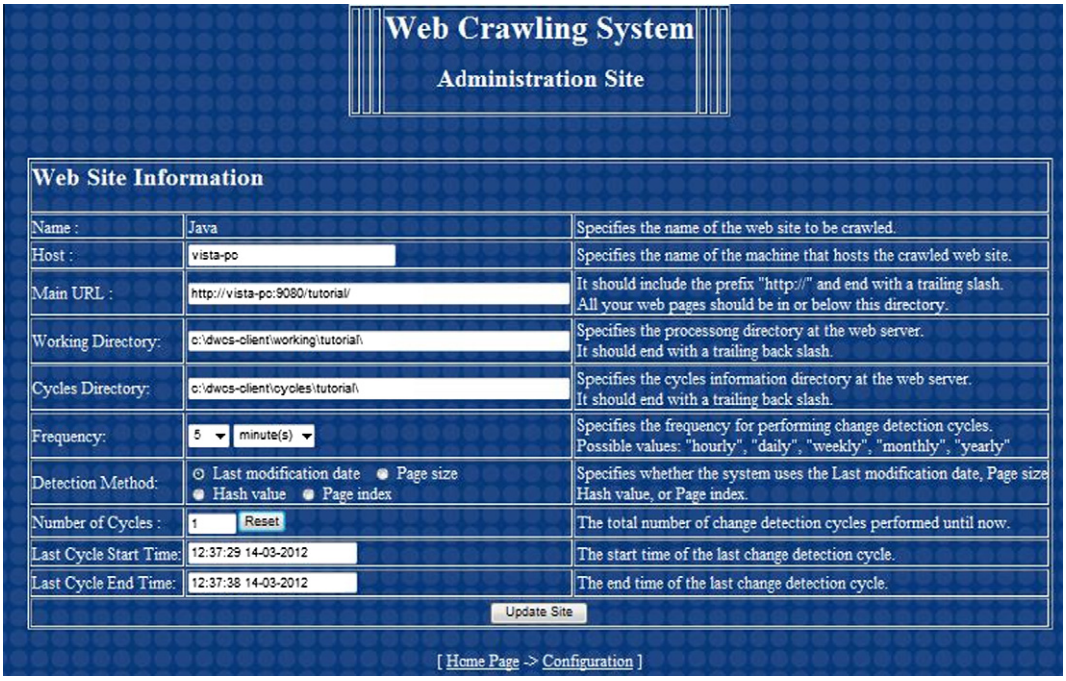


**Figure 4** Administration web page to register a web site at the system.

URL's list of all desired web servers. At the first crawling cycle, the MA creates and dispatches SAs to the desired web servers. The SA contacts the web server locally using HTTP request for all static pages and dynamic pages, indexes the crawled pages to create the document index which is sent to the MA. Also there is a copy of this document index saved at the web server for using later in the upcoming crawling cycles. At the further crawling cycle, MA dispatches a SA to a web server to start re-crawling process.

As we mentioned above, it is our attention toward content change of a web page, not the structural change. So it is important firstly before starting re-crawling process to determine if the web page which already crawled before has been changed or not, either the change structural or content. This filtration question saves web server's crawling time, indexing time, CPU cycles and memory. If the web page has not changed since the last crawling cycle, then no need to re-crawl and index it again. We determine that through the last modification date (LMD) of a web page which is saved in the meta-data of the web page. If the LMD has changed, then it indicates two probabilities, either a web page has structural change or content change. SA starts only re-crawling process for only crawled web pages with changed LMD. SA compares the web page's old document index which is saved on the web server from last crawling with the new one. If the indices of a web page are similar, then this web page has an insignificant change and hence no need to transmit its document index again over the network to the search engine. But if the indices are different, it means that a web page has actually changed in content and its document index should be transmitted over the

network to MA. MA receives the updated document index of a web page and replaces it with the old one in search engine's database. Afterwards the search engine creates the inverted index from the up-to-date document indices. The following information are collected and stored after each crawl during the experiment:

- Number of pages that were added/deleted/modified after the last crawl.
- The parameters responsible for change detection-last modification date, page size in bytes, hash value, keywords count (page index).
- The number of bytes retrieved directly by the crawler.

## 5. Experimental setup

A virtual environment has been setup to perform the experiments. There are two machines in our virtual environment, the first machine is Remote Site/Server (RS) that hosts a participating web site and the second one is the Search Engine. These machines have Pentium Dual-Core Intel processor clocked at 2.20 GHz with 2 GB of RAM and support both of java runtime environment and Java Agent Development Framework (JADE) [32] to develop multi-agent systems in compliance with the FIPA specifications. Both machines have 32-bit Windows operating system. These two machines are connected through high-speed LAN. As a data set for our experiments, 656 HTML web pages (about 4.95 MB total) are selected from ''Sun Java Tutorials'' website to evaluate the performance of our system. These web pages are selected, downloaded and stored on the RS. Fig. 4 shows the administration web page used by a web master to register his web site at DWCIS. The registration parameters include the main URL of the web site, host machine, the change detection technique and frequency.

Both stationary manager agent (as master agent) and mobile crawler agent (as slave agent) have been developed using JADE. The manager agent resides on Client server/Search Engine delegates the main URL's of participating web sites to the mobile crawler agents for crawling. The crawler designated for a website visited the Remote Site (RS) and analyzed the pages for modification after the last crawl and returned only the index of those that were actually modified. Fig. 5 shows JADE Remote Agent Management GUI that is used to manage and visualize the existing agents and agent containers. The agent container ''ManagerAgent-Container'' hosts the manager agent manager and the agent container ''Crawlers-Container'' hosts a pool of crawlers residing at the search engine side. In the example at Fig. 5, The agent crawler-0 was moved from the crawlers' pool to the agent container ''MyCrawler-Container'' hosted at the web server side to start a new crawling cycle. Terrier [33] is an open source Information Retrieval platform, containing common and modern statistical retrieval models, such as TF-IDF, BM25 and Language Modeling.

In particular, it provides state-of-the-art indexing and retrieval functionalities, and supports the rapid development and evaluation of large-scale retrieval applications. Terrier is implemented in Java and was used to create both document index and inverted index. We compressed the collection using WinZIP compression technique, and found that HTML documents are compressed approximately 70%.

Sitemaps are an easy way for webmasters to inform search engines about the pages on their websites that are available for crawling. By creating and submitting Sitemaps to search

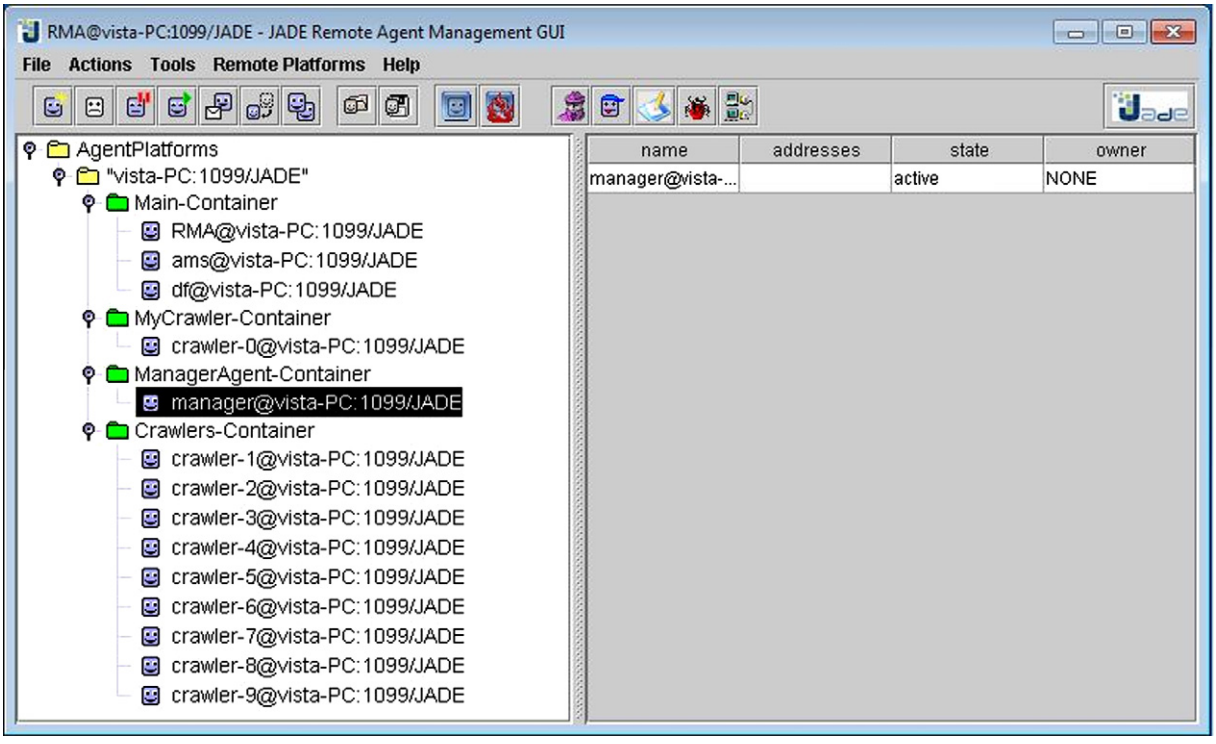


**Figure 5** JADE Remote Agent Management GUI.

engines, it is more likely to get better freshness and coverage in search engines. Google Sitemap Generator is a tool installed on the web server to generate the Sitemaps automatically [34].

## 6. Experimental results

The goal of our performance evaluation is firstly, to establish the superiority of our change detection technique based on document index over other studied change detection techniques, like Last modification date (LMD), page size by [6,18] and hash value by [13,14]. Secondly to establish the superiority of the distributed indexing approach against the currently centralized indexing approach, we measure performance in terms of the size of data transmitted across the network. As a data set for our experiments we used the Java programming tutorial as mentioned before in last section which is a set of 656 HTML pages (about 4.95 MB total). It is possible to reduce the network traffic further by compressing the pages before sending them to the Client Site (search engine). The HTML page can be compressed up to 30% of the actual size by using standard compressing tool such as WINZIP.

We compare our proposed distributed mobile crawler based on document index with other mobile crawlers using state-of-the-art change detection techniques. We have implemented the following crawler types:

- *Traditional crawler (TC):* It simulates a stationary crawler, running on the search engine side without any crawling computational load on the web server side. It downloads all web pages residing at the web server remotely using remote HTTP request. The search engine has the maximum computational burden as it has to process all web pages, create the document indices and create the inverted index used for query search.
- *Mobile crawler using LMD (MC1):* It is a migrating crawler based on Last modification date change detection technique. It migrates to the web server and filters the non-modified pages using the comparison of last modification dates at the time of page change detection. It sends back only to the search engine the modified web pages compressed.
- *Mobile crawler with page size (MC2):* It is a mobile crawler based on page size change detection technique. It migrates to the web server and filters the non-modified pages using the comparison of page sizes at the time of page change detection. It sends back only to the search engine the modified web pages compressed.
- *Mobile crawler with hash value (MC3):* It is a migrating crawler based on hash value change detection technique. It migrates to the web server and filters the non-modified pages using the comparison of hash values at the time of page change detection. It sends back only to the search engine the modified web pages compressed. A hash function is any algorithm or subroutine that maps large message of variable length, called keys, to smaller string of a fixed length. The values returned by a hash function are called hash values, hash codes, hash sums, checksums or simply hashes.
- *Mobile crawler with document index 1 (MC4):* It is a mobile crawler based on document index change detection technique. It migrates to the web server and filters the non-modified pages using the comparison of document indices at the time of page change detection. It sends back only to the search engine the modified web pages compressed.
- *Mobile crawler with document index 2 (MC5):* It is a mobile crawler based on document index change detection technique. It migrates to the web server and filters the non-modified pages using the comparison of document indices at the time of page change detection. It sends back only to the search engine the document indices of the modified web pages compressed. This crawler achieves the maximum reduction in the computational load on the search engine side because the document indices are already created. All what the search engine has to do afterwards is to collect the document indices from all the participating web sites and create the inverted index used for query-based search.

We evaluate the performance of the different above crawlers through six crawling cycles. For experiment, we assume that only one type of change will happen at each cycle. Table 1 shows the number of bytes transferred over the network between Search Engine and web Server caused by the six different crawlers with respect to the different crawling cycles. When mobile crawler is used, there is an overhead of 30 KB due to the mobile crawler itself, which is negligible compared to the data transferred by TC.

- *First cycle (initial crawling):* the proposed crawler has to download the compressed indices of all web pages hosted on the web server.
- *Second cycle (No change cycle):* No web pages have been changed since the first cycle. This cycle illustrates the advantage of mobile crawlers over traditional one.
- *Third cycle (pages deletion cycle):* In order to simulate the page deletion case, we randomly select 10 HTML web pages of total size 133 Kbytes and delete them from the web site.
- *Fourth cycle (pages addition cycle):* In order to simulate the page addition case, we added the 10 HTML web pages, deleted at the previous cycle, back to the web site.
- *Fifth cycle (Significant change cycle):* In order to simulate the case of page content change, we change the contents of the 10 HTML web pages, added at the fourth cycle. The change could be in the form of changing the page title or a sentence in a paragraph.
- *Sixth cycle (Non-Significant change cycle):* In order to simulate the case of insignificant page change, we change the 10 HTML web pages, added at the fourth cycle. The changes are divided into layout or structure change (e.g., changes in the position of HTML elements in the page, changes in comments), and attributes change (e.g., changes in fonts size and color, image size).

Tables 1–5 show the comparison between the traditional stationary crawler and the different types of mobile crawlers at different crawling cycles when 10, 20, 30, 40 and 50 HTML pages are changed, respectively. The traditional crawler retrieves all the HTML pages, of total size 4.95 MB, at the six cycles because it has no mechanism to process the web pages without downloading them to the search engine side.

- At the initial cycle, the migrating crawlers retrieve only 1.6 MB which is the compressed contents of the 656 web pages. We do not observe any difference between the three change detection techniques LMD, page size and hash value

**Table 1** Comparison between traditional crawler and different types of mobile crawlers at different crawling cycles when 10 randomly selected HTML pages are changed.

| Change detection technique | Initial crawling cycle | Further crawling cycle (10 pages) | | | | |
|---|---|---|---|---|---|---|
| | | No change | Significant change | | | Non-significant change |
| | | | Delete pages | Add pages | Content change | |
| Traditional (TC) | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB |
| *Mobile* | | | | | | |
| LMD (MCI) | 1.60 MB | 0 | 0 | 32.8 KB | 32.8 KB | 32.8 KB |
| Page size (MC2) | 1.60 MB | 0 | 0 | 32.8 KB | 32.8 KB | 32.8 KB |
| Hash value (MC3) | 1.60 MB | 0 | 0 | 32.8 KB | 32.8 KB | 32.8 KB |
| Index 1 (MC4) | 1.60 MB | 0 | 0 | 32.8 KB | 32.8 KB | 0 KB |
| Index: (MC5) | 616 KB | 0 | 0 | 10 KB | 10 KB | 0 KB |

**Table 2** Comparison between traditional crawler and different types of mobile crawlers at different crawling cycles when 20 HTML pages are changed.

| Change detection technique | Initial crawling cycle | Further crawling cycle (20 pages) | | | | |
|---|---|---|---|---|---|---|
| | | No change | Significant change | | | Non-significant change |
| | | | Delete pages | Add pages | Content change | |
| Traditional (TC) | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB |
| *Mobile* | | | | | | |
| LMD (MCI) | 1.60 MB | 0 | 0 | 47 KB | 47 KB | 47 KB |
| Page size (MC2) | 1.60 MB | 0 | 0 | 47 KB | 47 KB | 47 KB |
| Hash value (MC3) | 1.60 MB | 0 | 0 | 47 KB | 47 KB | 47 KB |
| Index 1 (MC4) | 1.60 MB | 0 | 0 | 47 KB | 47 KB | 0 KB |
| Index: (MC5) | 616 KB | 0 | 0 | 16.1 KB | 16.1 KB | 0 KB |

**Table 3** Comparison between traditional crawler and different types of mobile crawlers at different crawling cycles when 30 HTML pages are changed.

| Change detection technique | Initial crawling cycle | Further crawling cycle (30 pages) | | | | |
|---|---|---|---|---|---|---|
| | | No change | Significant change | | | Non-significant change |
| | | | Delete pages | Add pages | Content change | |
| Traditional (TC) | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB |
| *Mobile* | | | | | | |
| LMD (MCI) | 1.60 MB | 0 | 0 | 69.3 KB | 69.3 KB | 69.3 KB |
| Page size (MC2) | 1.60 MB | 0 | 0 | 69.3 KB | 69.3 KB | 69.3 KB |
| Hash value (MC3) | 1.60 MB | 0 | 0 | 69.3 KB | 69.3 KB | 69.3 KB |
| Index 1 (MC4) | 1.60 MB | 0 | 0 | 69.3 KB | 69.3 KB | 0 KB |
| Index (MC5) | 616 KB | 0 | 0 | 24.6 KB | 24.6 KB | 0 KB |

**Table 4** Comparison between traditional crawler and different types of mobile crawlers at different crawling cycles when 40 HTML pages are changed.

| Change detection technique | Initial crawling cycle | Further crawling cycle (40 pages) | | | | |
|---|---|---|---|---|---|---|
| | | No change | Significant change | | | Non-significant change |
| | | | Delete pages | Add pages | Content change | |
| Traditional (TC) | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB |
| *Mobile* | | | | | | |
| LMD (MCI) | 1.60 MB | 0 | 88.1 KB | 88.1 KB | 88.1 KB | 1.60 MB |
| Page size (MC2) | 1.60 MB | 0 | 88.1 KB | 88.1 KB | 88.1 KB | 1.60 MB |
| Hash value (MC3) | 1.60 MB | 0 | 88.1 KB | 88.1 KB | 88.1 KB | 1.60 MB |
| Index 1 (MC4) | 1.60 MB | 0 | 88.1 KB | 88.1 KB | 0 KB | 1.60 MB |
| Index 2 (MC5) | 616 KB | 0 | 32.1 KB | 32.1 KB | 0 KB | 616 KB |

**Table 5** Comparison between traditional crawler and different types of mobile crawlers at different crawling cycles when 50 HTML pages are changed.

| Change detection technique | Initial crawling cycle | Further crawling cycle (50 pages) | | | | |
|---|---|---|---|---|---|---|
| | | No change | Significant change | | | Non-significant change |
| | | | Delete pages | Add pages | Content change | |
| Traditional (TC) | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB | 4.95 MB |
| *Mobile* | | | | | | |
| LMD (MCI) | 1.60 MB | 0 | 0 | 110 KB | 110 KB | 110 KB |
| Page size (MC2) | 1.60 MB | 0 | 0 | 110 KB | 110 KB | 110 KB |
| Hash value (MC3) | 1.60 MB | 0 | 0 | 110 KB | 110 KB | 110 KB |
| Index 1 (MC4) | 1.60 MB | 0 | 0 | 110 KB | 110 KB | 0 KB |
| Index (MC5) | 616 KB | 0 | 0 | 41 KB | 41 KB | 0 KB |

where the corresponding migrating crawlers (MC1, MC2 and MC3 respectively) transferred the same amount of data which is 32.8 KB representing the size of the compressed contents of the ten changed web pages.

- In case of no change, using the migrating crawlers the network traffic can be reduced substantially compared to traditional crawler.
- In case of page deletion, the migrating crawlers returned back to the search engine side with an ACL message containing the names of the deleted pages so that the manager agent can discard them when creating the document indices or inverted index. The size of this ACL message is negligible compared to the size of the web contents. In case of significant content change, we do not observe difference between MC1, MC2, MC3 and MC4 (LMD, Page Size, Hash Value and Document Index 1 based crawlers); they transferred 32.8 KB which is the size of the ten HTML pages compressed;
- In case of non-significant change, both document index based crawlers MC4 and MC5 outperform the aforementioned crawlers as they ignore the changes and do not retrieve any bytes through the network.
- In case of page addition and significant change, document index based crawler MC5 performs the best as it retrieves only the compressed indices of the changed pages, with total size 10 KB, while it still ignores the insignificant changes as index-based crawler MC4. In other words, we can say that the index-based crawler MC5 filters the non-modified pages and sends lesser number of bytes to the Search Engine while maintaining the index up-to-date.

In the last experiment, we assume that only 10 web pages are changing. Fig. 6 shows the amount of data transferred when we increase the number of changed pages. We can see that the compressed index based crawler achieves the minimum data transmission rate in all the investigated cases. Fig. 6 shows the amount of data transferred by different crawler types when different number of pages are added or significantly changed. We can see that the compressed index based crawler MC5 achieves the minimum data transmission rate compared to the other non-index based crawlers MC1, MC2, MC3 and MC4 where it only transmits the compressed index of the modified pages.

Fig. 7 shows the amount of data transferred by different types of crawlers when different number of pages is insignificantly changed. We can see that both index-based mobile crawlers MC4 and MC5 achieve zero data transmission rate over the network while the other crawlers adopting other state-of-the-art change detection techniques do not differentiate between significant and insignificant page changes.

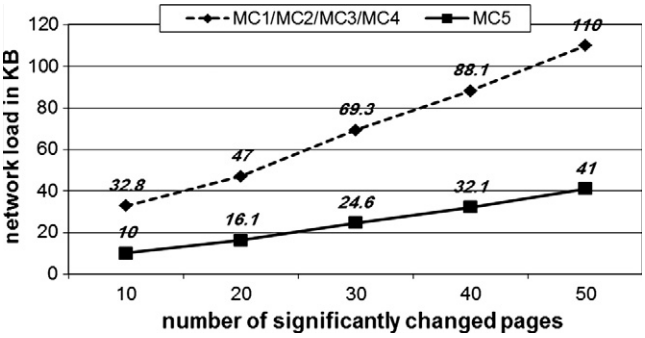


**Figure 6** Number of bytes transferred through the network when different number of pages are added or significantly changed.

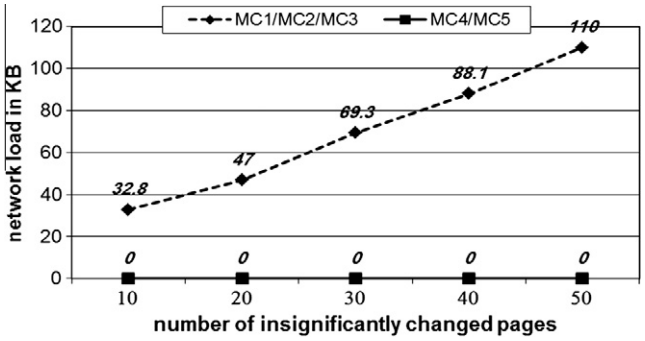


**Figure 7** Number of bytes transferred through the network when different number of pages are insignificantly changed.

## 7. Conclusion

This paper described a web page change detection technique based on document index, and distributed indexing using mobile agents in order to keep the search engine's index an up-to-date image to all web pages and other web resources hosted in web servers. The proposed change detection technique surpasses the other studied change detection techniques, like last modification date, page size and hash value. It considers significant web page changes which effectively reflect in the search

engine's index, while the other change detection techniques fail to differentiate between significant and insignificant changes. Of course the insignificant changes result in network overload and waste resources of both the search engine and the web server. Also the proposed distributed indexing approach surpasses the centralized approach of the current web indexing systems by distributing the data indexing on the web servers. In particular, using mobile agents we are able to perform remote operations such as data analysis, data indexing and data compression at the data source before the data is transmitted over the network. A virtual environment has been setup to perform the experiments. Experiments have shown that DWCIS has filtered out the pages which have not been modified significantly since last crawl. It has been found that DWCIS has reduced the network traffic after the document index creation and compression. It is also found that DWCIS has reduced the computational load on the search engine side compared to centralized traditional crawling (TC) and other mobile crawling techniques because the pages, which are not significantly modified, are not retrieved and the pages which are significantly modified, only their document indices are retrieved.

## References


[1] Sharma AK, Dixit A. Self-adjusting refresh time based architecture for incremental web crawler. Int J Comput Sci Network Secur (IJCSNS) 2008;8(12):349–54.

[2] Singh A, Singh K. Faster and efficient web crawling with parallel migrating web crawler Issues. Int J Comput Sci (IJCSI) 2010;7(3):28–32 [No. 11].

[3] Artail H, Abi-Aad M. An enhanced web page change detection approach based on limiting similarity computations to elements of same type. J Intell Inform Syst (JIIS) 2009;32(1):1–21.

[4] Yuan X, Harms J. An efficient scheme to remove crawler traffic from the internet. In: Proceedings of the 11th international conferences on computer communications and, networks; 2002. p. 90–5.

[5] Castillo C. Cooperation schemes between a web server and a web search engine. In: Proceedings of latin American conference on world wide web (LA-WEB) IEEE; 2003. p. 212–3.

[6] Bal S, Nath R. A novel mobile crawler system based on filtering off non-modified pages for reducing load on the network. Int Arab J Inform Technol 2011;8(1):272–9.

[7] Thati P, Chang P, Agha G. Crawlets: agents for high performance web search engines. In: Proceedings of the 5th IEEE conference on mobile agents; 2001. p. 119–34.

[8] Papapetrou O, Samaras G. Minimizing the network distance in distributed web crawling. In: Proc. of the 9th IFCIS international conference on cooperative information systems (CoopIS); 2004. p. 581–96.

[9] Cho J, Garcia-Molina H. Parallel crawlers. In: Proceedings of the 11th international conference on World Wide Web WWW; 2002. p. 124–35.

[10] Fiedler J, Hammer J. Using the web efficiently: mobile crawling. In: Proc. of the seventeenth annual international conference of the association of management on computer science; 1999. p. 324–9.

[11] Fiedler J, Hammer J. Using mobile crawlers to search the web efficiently. Int J Comput Inform Sci 2000;1:36–58.

[12] Brandman O, Cho J, Garcia-Molina H, Shivakumar N. Crawler-friendly web servers. In: Proceedings of the workshop on performance and architecture of web servers; 2000. p. 9–14.

[13] Yadav D, Sharma AK, Gupta JP. Topical web crawling using weighted anchor text and web page change detection techniques. J WSEAS Trans Inform Sci Appl Arch 2009;6(2):263–75.

[14] Khandagale HP, Halkarnikar PP. A novel approach for web page change detection system. Int J Comput Theory Eng 2010;2(3):1793–8201.

[15] Artail H, Abi-Aad M. An enhanced web page change detection approach based on limiting similarity computations to elements of same type. J Intell Information Syst (JIIS) 2009;32:1–21.

[16] Nath R, Bal S, Singh M. Load reducing techniques on the websites and other resources: a comparative study and future research directions. Int J Adv Res Comput Eng (IJARCE) 2007:39–49.

[17] Pahal N, Kumar S, Bhardwaj A, Chauhan N. Article: security on mobile agent based crawler. Int J Comput Appl (IJCA) 2010;1(14):5–11.

[18] Bal S, Nath R. A novel approach to filter non-modified pages at remote site without downloading during crawling. In: International conference on advances in recent technologies in communication and computing; 2009. p. 165–8.

[19] Singhal N, Agarwal RP, Dixit A, Sharma AK. Information retrieval from the web and application of migrating crawler. In: Proceedings of international conference on computational intelligence and communication systems; 2011. p. 480–3.

[20] Duhan N, Sharma AK. A framework for utilising usage trends in the crawling and indexing process of search engines. Int J Knowledge Web Intell 2011;2(4):272–91.

[21] Sharma AK, Mishra A, Singh V. An intelligent mobile- agent based scalable network management architecture for large-scale enterprise system. Int J Comput Networks Commun (IJCNC) 2012;4(1).

[22] Pinkerton B. Finding what people want: experiences with the webcrawler. In: The second international WWW conference Chicago, USA, October 17–20, 1994.

[23] Cho J, Garcia-Molina H. The evolution of the web and implications for an incremental crawler. In: Proceedings of the 26th international conference on very large data bases VLDB; 2000. p. 200–9.

[24] Sharma S, Sharma AK, Gupta JP. A novel architecture of a parallel web crawler. Int J Comput Appl 2011;4(1):2011.

[25] Yadav D, Sharma AK, Gupta JP. Parallel crawler architecture and web page change detection. J World Sci Eng Acad Soc (WSEAS) Trans Comput 2008;7(7):929–40.

[26] Sharma AK, Gupta JP, Agarwal DP. Parcahyd: an architecture of a parallel crawler based on augmented hypertext documents. Int J Adv Technol 2010;1(2).

[27] Cho J, Garcia-Molina H. Estimating frequency of change. Comput J ACM Trans Internet Technol 2003;3(3):256–90.

[28] Sharma DK, Sharma AK. Search engine: a backbone for information extraction in ICT scenario. Int J Inform Commun Technol Human Develop 2011;3(2):38–51.

[29] Agarwal A, Singh D, Pandey AK, Goel V. Design of a parallel migrating web crawler. Int J Adv Res Comput Sci Softw Eng (IJARCSSE) 2012;2(2).

[30] Aridor Y, Lange B. Agent design patterns: elements of agent application design. In: Proceedings of the second international conference on Autonomous agents; 1998. p. 108–15.

[31] Jun Su C, Ying Wu C. JADE implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring. J Appl Soft Comput 2011;11(1):315–25.

[32] The homepage of JADE is <http://www.jade.tilab.com/>.

[33] The homepage of Terrier is <http://www.terrier.org/>.

[34] The homepage of Google Sitemap Generator <http://www.code.google.com/p/googlesitemapgenerator/>.