Received May 20, 2020, accepted June 15, 2020, date of publication June 24, 2020, date of current version July 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3004756

SIMHAR - Smart Distributed Web Crawler for the Hidden Web Using SIM+Hash and Redis Server

SAWROOP KAUR AND G. GEETHA, (Member, IEEE)
Department of Research and Development, Lovely Professional University, Phagwara 144411, India

Corresponding author: G. Geetha (gitaskumar@yahoo.com)

ABSTRACT Developing a distributed web crawler obliges major engineering challenges, all of which are eventually associated to scale. To retain corpus of search engine and a reasonable state of freshness, the crawler must be distributed over multiple computers. In distributed crawling, crawling agents are given a task to fetch and download web pages. The number and heterogeneous structure of web pages are increasing rapidly. This made the performance a serious challenge to web crawler systems. In this paper, a distributed web crawler for the hidden web is proposed and implemented. It combines and integrates, scrapy framework and Redis server. Crawling is split into three stages-adaption, relevant source selection and underlying content extraction. The crawler accurately detects and submit the searchable forms. Duplication detection is based on hybrid technology using hash-maps of Redis and Sim+Hash. Redis server is also acting as a data store for a massive amount of web data so that the growth of hidden web databases is handled ensuring scalability.

INDEX TERMS Distributed crawlers, duplication detection, hidden web, web crawler.

I. INTRODUCTION

A web crawler is an automated software to browse the World Wide Web in an organized manner. By applying distributed computing technique to web crawling, the efficiency and effectiveness are improved in terms of time, cost, load balancing, and search quality etc. In distributed crawling, multiple agents work together for crawling the URLs and necessitate more complex approach than simple information curation. Mercator [1], Heritrix [2], Nutch [3], Scrapy, JSpider, HTTrack [4], YaCy [5], etc. are some of the distributed web crawlers in use. The hidden web is a part of the web that is masked behind the HTML forms and is not generally accessed by web crawlers. For this purpose, we need a crawler that can find all webpages with in searchable forms and can also fill and submit the form automatically without any manual intervention. EFFC [6], Adaptive crawlers [7], FFC [8] are some of the web crawlers that can handle the hidden web. To access the hidden web, there are two approaches namely virtual integration and web surfacing.

II. LITERATURE REVIEW

The literature review is organized into three parts. The first part review the distributed web crawlers, followed by focused and hidden web crawlers.

The associate editor coordinating the review of this manuscript and approving it for publication was Senthil Kumar.

Zhou *et al.* [9] have designed a distributed vertical crawler using crawling template-based periodic strategy. The domain of crawling is internet forums and performance of the same has been measured in terms of the number of URLs that has been processed. Results have shown that distributed crawling has gathered a greater number of URLs than single vertical crawler, when compared. Gao *et al.* [10] have designed geographically distributed web crawler and tested on various crawling strategies. Out of which, URL based and extended anchor text-based have given the favorable performance. Yu *et al.* [11] have presented cluster-based distributed crawler implemented as a data server. This crawler is shopping product based so do the feature extraction. Web server is presented with processed data. Scalability is provided using a Hadoop platform. Hbase is used to store huge data. The assumption for load balancing is that when all the nodes finish their crawling task at the same time. Performance of crawler is compared with Nutch crawler. With 8 crawling nodes between 3500-4000 pages are crawled per minute. Feng Ye et al [12] have implemented distributed crawler based on Apache Flink. On the cluster, Redis and other databases are deployed to store the web pages that are crawled. Scrapy is selected as an underlying crawling framework. Duplication detection is employed by combining the bloom filter with Redis. Performance is measured in terms of crawled pages and execution time. The crawler has managed to crawl 20000 pages in seven hours. CPU utilization rate even at the fourth hour is

TABLE 1. **Comparison of reported distributed web crawlers.**

REFERENCE	DG	DF	DV	MAX NO OF NODES	CT	PAGES/URL	DS	MT	CLASSIFICATION	CPU-U	T
[11]	-	-	✓	3	-	26136	361.50	-	-	-	-
[12]		-		-	60 secs	4000	-	-	-	-	-
[13]	✓	-		-	7HRS	7000	-	-	-	35%	-
[14]	✓	-		50 rounds	6000	-	-	-	-	-	10^6 -10^7
[15]	✓	-		5VM	153.04	8452	-	-	-	-	-
[16]	-	✓		-	4 hours	-	-	2000 at 550 Mbps	-	70	-
[17]	-	✓	✓	-	-	-	-	-	99.4	-	-

less than 35% as compared to a single crawler. Duplication detection is compared with bloom filter, link list, hashmap and treemap with bloom filter giving promising results. The number of fetched pages increases to 7000 when system used Mesos/Marathon platform.

UniCrawl, a geographical distributed web crawler worked upon by Moftah and Abuelenin [13], have yielded 5000 new URLs in 50 crawling rounds and yielding throughput between 10^6 to 10^7 for 6000 seconds. M. ElAraby *et al.* [14] have developed a dynamic web crawler as a service. Each stage of this architecture worked as a separate service and deals with its load. So, scalability is also based on individual stages. The whole system does not need to be scaled. Along with being dynamic, this architecture is customizable and provided standalone services using elastic computing. The system has used Amazon RDS service. Performance is compared for fetched pages vs time graph. This crawler can fetch more than 250 pages in less than 400000 seconds. Then using 5 virtual machines 300 pages are crawled in 153.04 seconds. With the same configuration number of discovered URLs are 8452. This system has also worked on discovering new domains from newly discovered URLs. Comparison is made between response time for multithreaded crawler and virtual machines. For 300 pages, the response time of multithreaded crawler is 142132.4 and virtual machines on cloud computing are 512159.8.

According to Achsan and Wibowo [15] infinite threads curtail the performance of web crawler. The system has divided crawling based on the heuristic that the large site are crawler before smaller size sites. Results are compared for CPU and memory utilisation. For 2000 threads CPU utilisation is 70% at 550 Mbps bandwidth. Choosing a suitable approach to divide the Web is the main issue in parallel crawlers.

Achsan and Wibowo [15] have worked on politeness property. Bosnjak *et al.* [16] proposed continuous and fault-tolerant web crawler called Twitter Echo. This crawler continuously extracts data from twitter like communities. Performance is measured in terms of classification accuracy with 99.4% of the highest classification accuracy for non-Portuguese sites.

Raj *et al.* [16] have developed a platform-independent distributed crawler that can handle AJAX-based applications. They have also supported the breadth-first search for complete coverage. Performance is compared up to 64 active threads to crawl two-page application and medium sized application.

Xu *et al.* [17] have implemented distributed crawler based on Hadoop and P2P. All the files are stored and shared from in the distributed file system. Performance is measured as time to crawl vs nodes.

DG-	Distributed General
DF-	Distributed focused
CT-	crawling time
DS-	Downloading speed
MT –	Maximum threads
CPU-U –	CPU utilisation
T –	throughput

Table 1, shows the comparison of existing distributed web crawlers based on their performance measures.

A. HIDDEN WEB CRAWLING

To crawl the data hidden behind the web forms, the following steps are performed.

1) AUTOMATED HIDDEN WEB ENTRY POINT DISCOVERY

The deep web site can be discovered in two ways: either using heuristic or machine learning. Madhavan *et al.* [18] used heuristics to discover form tag and other features of forms that includes- presence of a number of a text box. Other way is to use heuristic to discard forms with short input as implemented used by Bergholz and Chidlovskii [19]. While Cope *et al.* [20] and Barbosa *et al.* [21] have applied machine-learning algorithm to classify forms to find entry to the hidden web.

2) FORM MODELLING

After entry to the deep web, the next step is form modelling. that includes the identification of the type of classification. Forms can also be classified studies based on pre-query or post query. In the post query case response page is a source of classification. Feature of each form is also the source of classification.

3) QUERY SELECTION

Kashyap *et al.* [22] have used the concept of static hierarchy for query selection. BioNav has been used for the hierarchies. The performance measure is based on the overall cost of the queries. The aim is to retrieve a greater number of records. Chen *et al.* [23] have worked on both the content and the structure of the form for queries as well as databases. The quality of the query is measured in term of difficulty over the database. This model estimates the number of queries compulsory to retrieve the whole content of the hidden web site. Performance is measured in terms of correlation of average precision. Madhavan *et al.* [24] have proved that the load on the system increases by increasing the number of submissions. As some queries generate duplicate results. So, the query selection technique should have the goal of "minimize the number of queries and maximize the accurate response". Barbosa and Freire [25] proposed model for unsupervised keyword selection. This model starts with keywords extracted from the form page. First, most frequent words are calculated, and submission is repeated until maximum results are obtained. Performance is measured in terms of the effectiveness of the technique with and without using a wrapper. Ntoulas *et al.* [26] proposed a technique in which for submission is done with the user-provided keyword. It also extracts keywords from the response pages. The keywords with higher informativeness are selected., which is calculated as their accumulated frequency.

4) CRAWLING PATH LEARNING

Searchable forms can be reached using a path learning process. Relevant page with correct response can be generated by following the pages in an order. Based on path learning crawlers can be categorized as blind crawlers, with aimless but perpetual download capacity and focused crawlers [28], [29] that works on a fabricated path and reach the desired web page. The proposed crawler is based on the focused crawlers. Overall performance check of the crawlers is based on their access to the hidden websites and to measure the finding capacity and accuracy of the desired forms. Hidden web crawling neither have standard dataset nor comparison framework and testing environment to compare features of the techniques. So the reported research is compared using different techniques used in hidden web crawling.

B. FOCUSED CRAWLERS

Focused web crawlers play an important role in creating and maintaining subject-specific web collections. Application of focused crawlers includes search engines, digital libraries, specialized information extraction and text classification, high-quality result page, minimizing the time, space and network bandwidth. The goal of a focused crawler is to retrieve maximum relevant pages. Focused web crawlers like general web crawler have same components called:

1. Fetcher or downloader which fetches the web page and retrieves its contents.
2. Frontier that stores the URLs of unvisited websites along with the visited one, for extraction of further information

In addition to these three components, focused web crawler has a topic-specific crawling model, relevance estimation and ranking module. Focused crawler first collects some URLs as seed sites. From these URLs, a crawler begins its crawling process and give results in the form of webpages crawled.

From the above literature, we conclude that distributed crawlers for the hidden web are limited and they face performance issues in terms of scalability, duplication, and unable to support frequent change in the underlying technology of web pages. To address the above-mentioned challenges, focused distributed web crawler is developed that can handle duplication and scalability. Duplication detection is based on hybrid technology using hash-maps of Redis and Sim+Hash. Redis server is also acting as a data store for a massive amount of web data so that the growth of hidden web databases is handled ensuring scalability.

Plethora of literature is available on the information retrieval methods, however there is no refined information about the working and liability of distributed web crawlers and their role in bringing hidden web data to light. Some emerging research in this area used MapReduce to compute term frequency etc. As opposed to general web crawling, hidden web crawling requires a complex approach to parse, process and extract information from the hidden websites. And similarly, the process of distribution in hidden web crawling is equally challenging. Performance of crawler is highly influenced by architecture and techniques of crawling. From the literature review, it is found that Distribution can be implemented to covercome the drawbacks such as scalability, duplication, and inability to support frequent changes in the underlying technology of web pages. The crawlers working in a methodical approach can effectively touch specific topics. As per the literature there isn't any focused distributed web based crawler designed to uncover hidden data.

III. THE PROPOSED ARCHITECTURE

The proposed architecture work in three stages.

1. URL adaption and classification: Frontier is initialized in this phase, followed by parameter learning, ranking and domain classification.
2. Relevant source selection: When frontier encountered a URL, all the links are extracted in link frontier, and in fetched link frontier.
3. Underlying content extraction: While in the third stage the form structure is extracted to fill and submit the forms.

This system has implemented the frontier as a queue from which URLs are taken out for further processing. The frontier starts from the seed URLs. We have implemented three queues as frontiers. The frontier for seed URLs consists of URLs from the directory. The frontier for links consists of URLs extracted from the seed URLs. The frontier for fetched links consists of URLs from links. The frontier depletes very

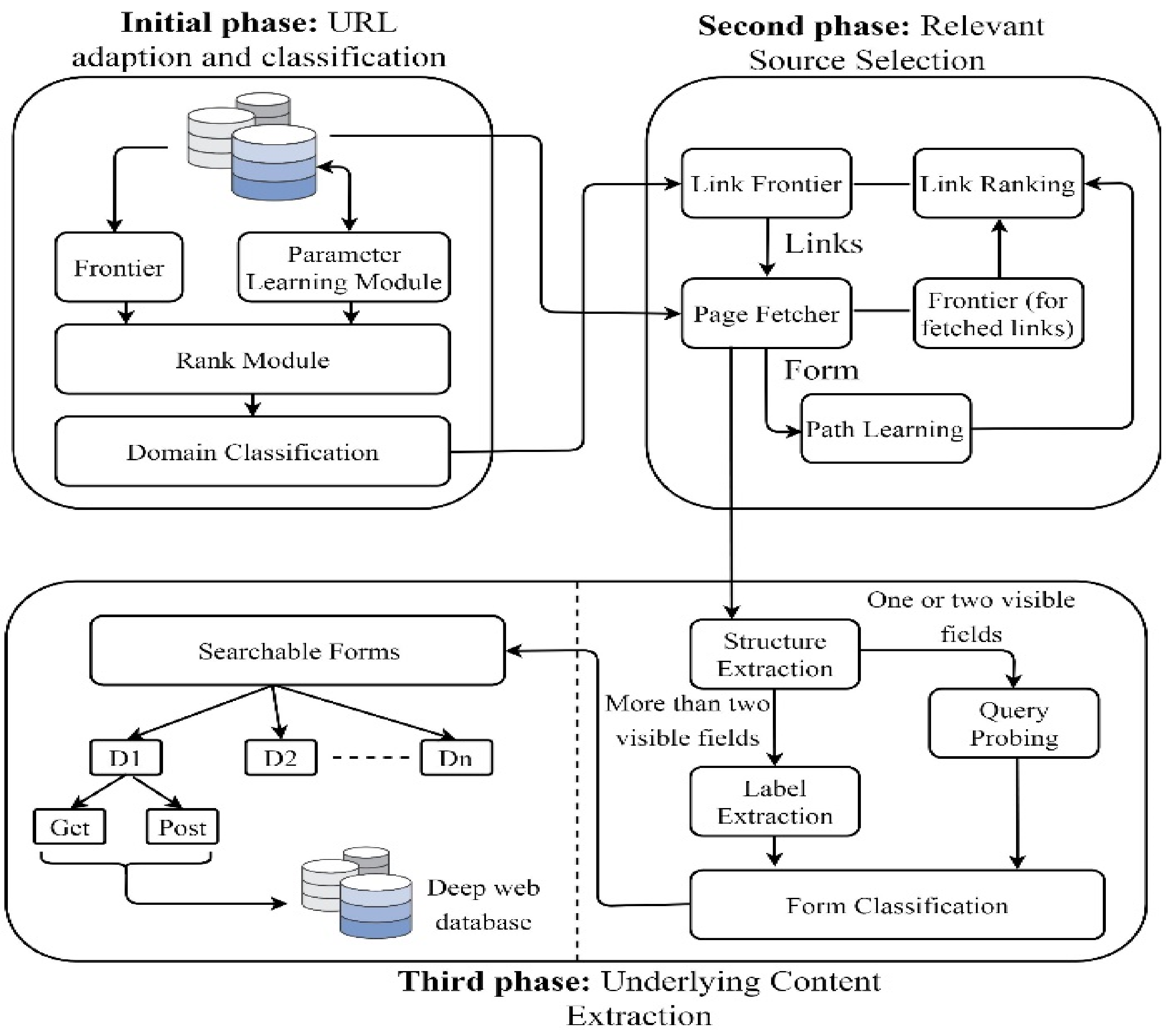
Initial phase: URL adaption and classification
Frontier
Parameter Learning Module
Rank Module
Domain Classification
Second phase: Relevant Source Selection
Link Frontier
Link Ranking
Links
Page Fetcher
Frontier (for fetched links)
Form
Path Learning
Searchable Forms
D1
D2
Dn
Get
Post
Deep web database
Structure Extraction
One or two visible fields
Query Probing
More than two visible fields
Label Extraction
Form Classification
Third phase: Underlying Content Extraction

FIGURE 1. Architecture of SIMHAR as a single entity focused crawler.

easily. So as the frontier for seed URLs will have a scarcity of URLs, frontier for links will be used. A webpage can have multiple hyperlinks but not all are relevant. Aim of a web crawler is to fetch maximum deep websites by minimizing the visited URLs. Following figure 1 show three phases of the crawler. All the stages are interconnected with each other. As the URL is extracted from the frontier, the next step is pre-processing of URLs.

A. PRE-PROCESSING OF URLS

The baseline components of URLs are extracted. These are host, extension, documents, path etc). from all the components URL, path, anchor and text around anchor are fed to feature vector. The system has implemented python NLTK for stemming, stop word removal and tokenization. Now all the segmented words are fed to feature vector.

Feature space for the hidden website is defined as:

FD = [URL, anchor, text around anchor].

Feature space for links of the hidden website is defined as:

FL = [path, anchor, text].

Path of the URL is learned to reach the exact location of the form. special symbol related to the path is the forward-slash (/). Path of the URL is found after the hostname. Anchors are helpful in internal navigation in URL. And we need to find the internal links as well.

B. WEIGHT CALCULATION OF TERMS

Based on feature vector construction, we have to compute the weight of term corresponding to its occurrence in URL(U), anchor(A), text around the anchor (T) and path (P).

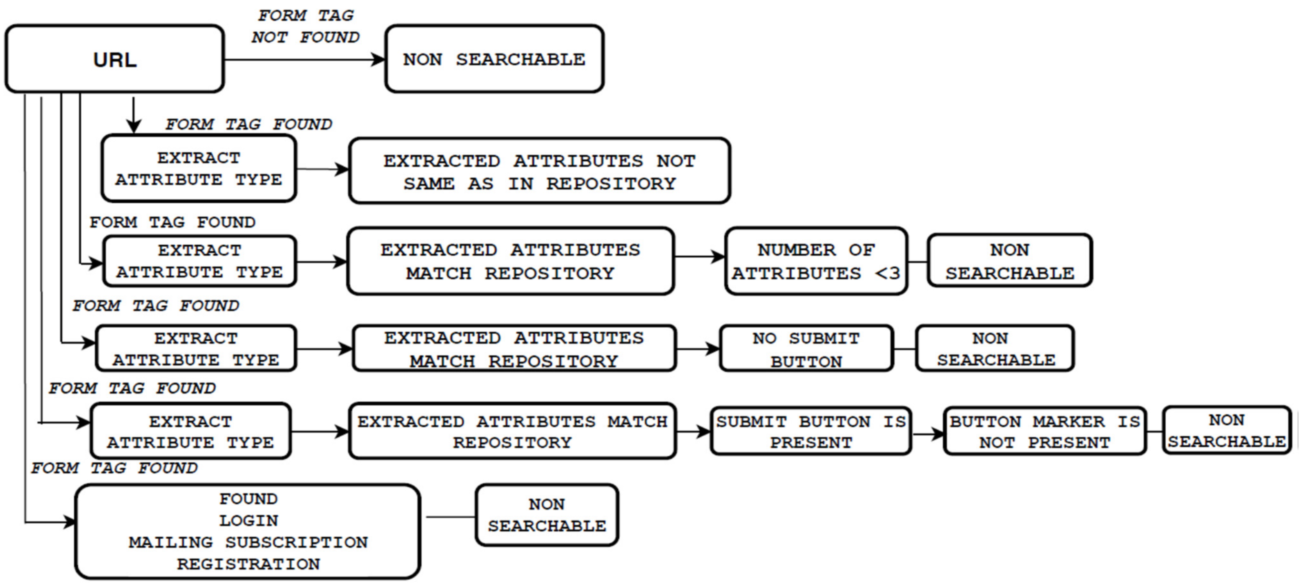

FIGURE 2. Rejection framework for forms.

Term frequency of term T_i in U, A, T and P and is defined as:

$$t_i f_i = \alpha \times t_i f_i + \beta \times t_i f_i + \gamma \times t_i f_i + \delta \times t_i f_i \quad (1)$$

where α, β, γ, and δ are the weight coefficient. Ig is the information gain of terms.

$$\frac{w_{ij} = t_i f_i \times \dot{i} df_j \times Ig}{\sqrt{\sum_{N=1}^{N}} (t_i f_i \times \dot{i} df_j) 2} \quad (2)$$

It is proved in [12] that outcome of the tf-idf alone is an inappropriate distribution of feature vector. In their approach, information is combined with the segmentation of page in a major four sections. In our approach weights are based on URLs and associated terms. After term weighting similarity is computed. Similarity(S) is computed between the already discovered URL and newly discovered URL. The similarity is required in the ranking section. The similarity is computed as.

$$S = \mathrm{sim}(U, U_{new}) + \mathrm{sim}(A, A_{new}) + \mathrm{sim}(T, T_{new}) \quad (3)$$

Similarity has a different meaning concerning each step-in web crawling. The crawler has to work on finding similar URLs so that it can prevent similar data retrieval. It also needs to find similar content for top k queries, as well as text with semantic similarity.

After pre-processing, the system has a list (k) of more than 50k keywords. Now using the similarity model (V), the system read the reference file. In the next step, the elements are removed from the list (k) one by one. The similarity is computed between two lists. For example, the flight is a word in list (k) and it has a close match in (V). If cosine similarity is 1, it means an exact match is found. Close match results are used for queries during repository generation in form submission. Crawler removes duplicate URLs from the frontier using Redis hash maps. The websites have a complex relationship with each other. Same URL can be found on multiple websites. Downloading the same URLs multiple times is a waste of resources. Redis database includes a de-duplication set. So that's why the unique fingerprint for each request is calculated first. Fingerprints are put in this set. All the repeated requests are removed here. Simhash and Levin [30] is combined with Redis for improvement in results.

C. LEARNING

Feature construction is explained in section A. The crawler is adaptive in nature, results from the first run are used in successive runs. Following steps are performed in the learning algorithm.

1. A new website (X) is encountered, extract [U, A, T].
2. For each URL, arrange the frontier using a similarity model with respect to [U, A, T].
3. Extract the links from X.
4. Extracted links are saved in the link queue. The link queue is ordered using the similarity model with respect to [P, A, T].
5. Check for searchable forms by following rejection rules as shown in Figure 2.
6. If the form is searchable, extract path, anchor and text.
7. Update the information in parameter learning module in stage 1, and link ranking in stage 2. And new features are reflected in these two modules.
8. Stop, if crawler has reached the threshold of 0.8, i.e 80 new URLs and 0.1, i.e 100 new forms at depth one. Follow steps 1-8 for depth 2 and depth 3.

D. RANKING

Aim of ranking in hidden web crawling is to extract top n documents for the queries. The cost is expected to be the least for this work. We have adopted the formula for ranking from [31]. But our reward function (€) is based on a number of out-links, site similarity and term weighting. Let SF be the frequency of out-links.

$$\mathrm{SF} = \sum \mathrm{I_i} \tag{4}$$

$$\begin{aligned} \mathrm{I} &= 0, \quad \text{site has not appeared,} \\ \mathrm{I} &= 1, \quad \text{if it has appeared} \end{aligned}$$

$$€ = w_{ij} + S + SF \tag{5}$$

$$(r_j) = (1 - w).\delta_j + w.ranking\ reward(€)/c_j \tag{6}$$

w is the weight of balancing $€$ *and* c_j. δ_j is the number of new documents. Computation of r_j shows the similarity of € and returned documents. If the value of € is closer to 0, it means that returned value is more similar to the already seen document i.e the new URL is similar to the already discovered URL. c_j *is a* function of network communication and bandwidth consumption.

E. DOMAIN CLASSIFICATION

Domain classification is based on topical relevance of the site and the home page. As the new URL is received its homepage content is parsed and feature vector is constructed. The resulting vector is fed to the classifier to check relevancy. The crawler gets the URLs and the request is sent to a server to fetch the page. The crawler will first check for the presence of a search interface. The forms are two types defined as follows:

- Searchable form: "Webform is called searchable form if it is capable of submitting a query to an online database which in turn, return the results of a query."
- Non-Searchable form: "The forms, for example, login registration, mailing list subscriptions forms, and so on are called non-searchable forms. These forms do not represent database queries.

A crawler encounter number of types of web pages, but not every web page is searchable. So following rules are designed to help crawler decide whether the encountered form is searchable or not. After these rules are applied, the crawler has a set of URLs which have < form> tag as well as the property of being searchable. On being provided with suitable values, these forms will retrieve the data from the associated database.

Rule1: If crawler do not find any < form> tag, consider this a non-searchable form.

Rule2: If crawler found the < form> tag. Then extract the attribute type. If the attribute type is not in repository call it a non-searchable page.

Rule 3: If crawler found the < form> tag, and extracted attribute type matched in a repository. But the attributes < 3, consider this page non-searchable.

Rule 4: If the number of attributes is >3, but the submit button is not found, consider this page as non-searchable.

Rule 5: If there exists < form> tag, and attributes are similar to the repository, and submit button is also there. But button marker is not present then consider this page as non-searchable.

Rule 6: If there exists < form> tag, and attributes are similar to the repository, submit button and button marker is are present. It is a searchable form.

Rule 7: If there exists < form> tag, but crawler found login, then this is non-searchable.

Rule 8: if there exist < form> tag, but crawler found registration, consider this page as non-searchable.

Rule 9: if there exist < form> tag, but crawler found subscribe, then consider this page as non-searchable.

Rule 10: If there exist < form> tag, but crawler found mailing list subscription, then consider this page as non-searchable.

Figure 2 shows the diagrammatic view of the above-mentioned rules.

F. FORM STRUCTURE EXTRACTION

After the webpage with form is found, the content of the form is extracted. Search forms have controls that a human can easily fill and submit. If a crawler has to fill the forms automatically it has to have a set of resources to automatically fill and submit the forms with suitable values. A task-specific database called repository, is initialized at the launch. This database contains the set of values for filling the forms, created by parsing the form as shown in table [2]. And form element table is created with a control element type, label and domain values. The crawler will adaptively learn filling values with associated forms. When the first run of the crawler is completed, the parsed values will be analyzed to collect data. Form submission is of two types: post-form submission and get form submission. This crawler work on both type of submissions. After the form is submitted crawler got the response status. Response status is either a valid page or no page found code. Following steps are performed during form parsing:

- Using Request library of python, HTTP GET request is sent to URL of a webpage.
- The response of HTTP request is HTML content of a webpage.
- Data is fetched and parsed using Beautiful soup.
- HTML tags and their attributes are analysed.
- Data is output in CSV file.

G. FORM AND RESPONSE ANALYSIS

Forms have multiple control elements. It could be of any type:

- Text: This area of a form can be edited with multiple lines of words.
- Input: This editable area has following attributes types-type as text, Submit, checkbox and radio button
- Select: Select has two options like drop-down list box and multi-choice list box.

Two heuristics are implemented based on visible fields. If the number of visible fields is one or two -forms are classified using query probing, label extraction otherwise. As explained in [32] form submission include problems like 404 error page, duplicate information, and sometimes all information is retrieved in single submission otherwise

TABLE 2. Contents of repository.

Control Element (Visible Fields)	Label	Domain	Type Of Domain	Size	Status
Submit	Search	Submit	Infinite	More than 3 kb	VR
Radio	Flight trip	Round trip One-way trip	Bounded	More than 3 kb	VR
Select	From, to	Name of place (eg: Delhi to America)	Bounded	More than 3 kb	VR

multiple submission are required. The crawler has also faced these problems.

H. QUERY PROBING

Aim of query probing is to develop a set of queries for each class. On submitting these queries crawler will retrieve the same documents for that category. Our approach is similar [33], but we have implemented hierarchal classification, and the system can expand to the number of classes as it crawls more URLs. Currently, classes are based on seed dataset.

I. FORM SUBMISSION

Two related techniques with form submission are:

HTTP POST: In post query technique, forms are submitted with (name, value) tuple. This pair is sent encoded in the body of the request. Query probing is implemented in post method technique.

HTTP GET: In get query technique, forms submission takes place with giving (name, value) pairs in URL. The pre-query technique is implemented in get method technique. In get method URL has three symbols a question mark (?), equals to (=) and ampersand (&). (?) differentiates encoded (name, value) from the base URL and action path. (=),(&) separates the field name and field value.

J. STOPPING CRITERIA

Exhaustive crawling is waste or resources. This system has implemented following stopping criteria.

- Maximum depth of crawl: the crawler will stop following the link when the depth of three is reached. It is proved in [23] that most of the deep web pages are found till depth 3. While at each depth maximum number of pages to be crawl is 100.
- At any depth maximum number of forms to be found is 100 or less than a hundred.
- If the crawler is at depth 1, it has crawled 50 pages, but no searchable form is found, it will directly move to next depth. And the same rule is followed at depth 2. Suppose if at depth 2, 50 pages are crawled and no searchable form is found. The crawler will fetch new link from URL.

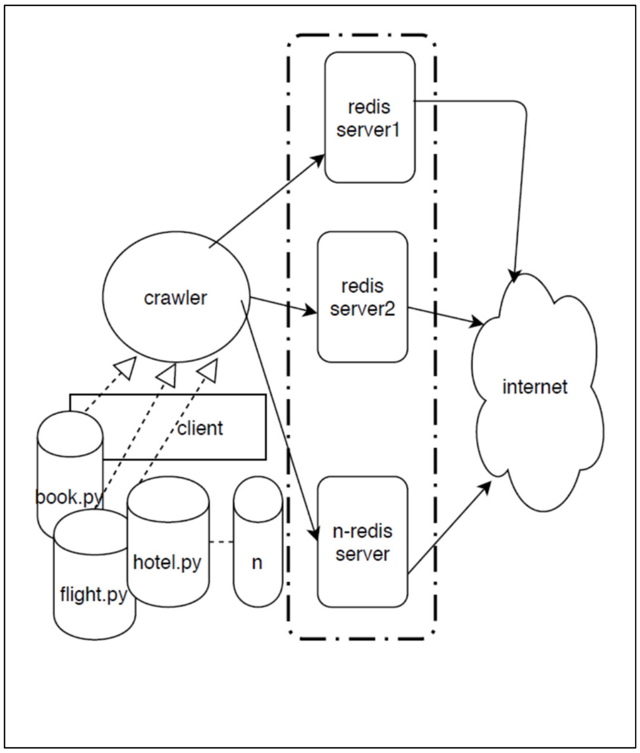

FIGURE 3. Distribution of SIMHAR based on Redis server.

K. ASSUMPTIONS AND THRESHOLDS

- The size of the frontier should not decrease below 100 URLs at a time, as the number decreases, it will crawl URLs from link frontier and fetched link frontier.
- Learning threshold is 80 new sites and 100 new searchable forms.
- URLs are picked out from a crawler using first in first out order.

L. DISTRIBUTION

The above architecture (figure 1) describes the working of a single entity of focused crawler for the hidden web. The Redis server is implemented as shared storage for URLs. Redis store information in cache, unlike databases, that is why information access is faster. The proposed crawler is developed in Python. Scrapy is an application framework. Scrapy helps to extract web pages and structural data. For distributed crawling, Scrapy and Redis are integrated to implement more than one server. A crawler is implemented with breadth-first search per host. Data can be extracted either by using API of a website or by extracting information by accessing the webpage. To create a tree structure of HTML data html5lib parser library is used. To navigate through parse tree beautiful soup is used. It can pull any type of data. The following figure 3, explained distribution using multiple Redis to make

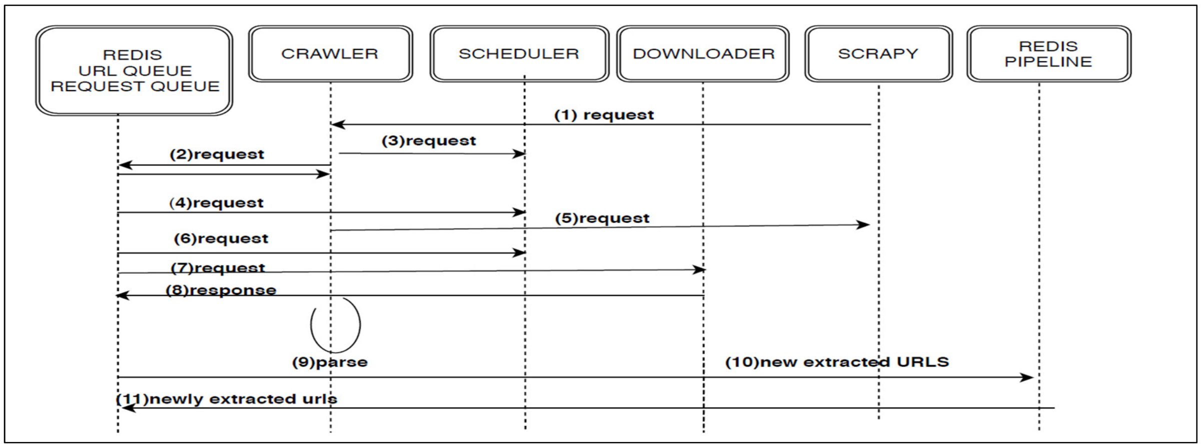

FIGURE 4. Job scheduling in proposed crawler.

it fault-tolerant. Also the links captured from the web must be shared with all the crawlers.

M. JOB SCHEDULING

Because scrapy has no mechanism for link sharing even though it has schedular, so the URLs are shared from memory of crawler. Figure 4 show the step wise implementation of job scheduling. The task of job scheduler is to prevent overloading the websites.

Web pages are popped out from the frontier first-in-first-out way. We have considered this as our baseline assumption. Crawling in breadth-first fashion is implemented as URL and server-based. It is proved in [34] that it yields promising results. Following steps are performed in Job Scheduling.

Step 1. To start crawling, scrapy send schedule request to message to a crawler.

Step2. As crawler receives the request it starts crawling. From the Redis URL queue, a URL is selected and sent as a request to schedular.

Step 3. Schedular receives a request of URL, it forward this to Redis (request queue), and then again contact (request scheduled) is made with scrapy.

Step 4, 5. Now the associated webpage is to be downloaded, for this request is popped from the top of a request queue, and downloader on receiving the request, download the request page.

Step 6, 7. Downloader after getting the contents of a page, submit it page to the crawler.

Step 8, 9. Crawler parses the webpage, collect the new URLs and send a new list of URLs to Redis pipeline. Redis pipeline send these URLs to Redis queue. One another advantage of.

IV. CONFIGURATION

The system hardware environment includes: CPU is Intel®, Core^{TM} C5-7200@ 2.50 GHZ 2.70GHZ, with installed RAM-12.0 GB, and Redis 3.0.509. the crawler is implemented in python. The internet speed during the experiment was 50-100mbps.

V. EVALUATION

This crawler has to first check if the page belongs to hidden web or not, by following the rules in Figure 2. After the seed database URLs are checked for $<$ form$>$ tag, the crawler has to pull the contents of the webpage using the URL. The request library help make use of HTTP with in the python program. Beautiful Soup can extract any type of data from a webpage. After the HTML Markup's are removed page is saved for further processing. Beautiful soup is combined with urllib3 to work with web pages. Other way is to download a copy of webpage then use it locally. Beautiful soup has feature called ''prettify'', in which all the unnecessary tags can be dropped. We have selected 6 domains from the dataset. This dataset contains more than 260000 associated URLs.

Initially the DMOZ dataset is used. The performance of the classifier is measured using confusion matrix. Rows of confusion matrix denote actual class, while column indicate classes predicted by SVM and knn classifiers. We have computed accuracy for each class. Average of each class denote the performance of the classifier. The performance metrics are precision, recall and f1. Precision is classification of portion of webpages that are relevant to the class. It means how correct the system is to reject the web pages that are not relevant. Recall is how correctly classifier can find relevant document. In Redis multiple jobs are separated using unique keys. So, jobs are not mixed.

Table 3 shows the description of status codes. Web forms cannot be submitted for these codes. Table 4 shows the values of precision, recall and accuracy using support vector machine for status codes mentioned in table 3. This shows how accurately system has detected the forms which cannot be submitted.

Table 5 shows the confusion matrix for correctly submitted forms. Table 6 and table 7 show the submission accuracy results using support vector machine and K- nearest neighbor

TABLE 3. Status and their description.

Status code	Description
200	Asynchronous response
400	Bad request error
404	Page not found
413	Payload too large -Request entity is large
414	Payload too large -URI too long
500	Internal server error
503	Service unavailable error

TABLE 4. Precision, recall and F1 score for status code, using SVM.

STATUS CODE	PRECISION	RECALL	F1-SCORE	SUPPORT
200	0.65	0.95	0.79	875
400	0.00	0.00	0.00	72
404	0.56	0.20	0.30	168
413	0.00	0.00	0.00	60
414	0.00	0.00	0.00	16
500	0.00	0.00	0.00	13
503	0.97	0.85	0.91	1150
accuracy			0.89	4761
Macro avg	0.40	0.38	0.37	4761
Weighted avg	0.88	0.89	0.88	4761

TABLE 5. Confusion matrix for correctly submitted web pages.

Actual Class							Total
Flight	2369	0	0	0	0	0	2369
Book	0	7790	3	26	0	67	875
Hotel	0	56	23	0	0	14	93
Product	0	105	0	55	0	8	168
Music	0	46	0	0	0	0	46
Auto	0	11	0	2	0	7	20
Predicted Class	Flight	Book	Hotel	Product	Music	Auto	10698

TABLE 6. Precision, recall and F1 score for correctly submitted code using SVM.

CLASS	PRECISION	RECALL	F1 SCORE
FLIGHT	1.0	1.0	1.0
BOOK	0.99	0.97	0.98
HOTEL	0.97	0.88	0.39
PRODUCT	0.25	0.68	0.44
MUSIC	0.33	0.0	0.0
AUTO	0.39	0.73	0.12

TABLE 7. Precision, recall and F1 score for correctly submitted code using KNN.

CLASS	PRECISION	RECALL	F1 SCORE
FLIGHT	0.70	0.89	0.79
BOOK	0.79	0.25	0.38
HOTEL	0.66	0.33	0.44
PRODUCT	0.00	0.00	0.00
MUSIC	0.00	0.00	0.00
AUTO	0.92	0.91	0.91

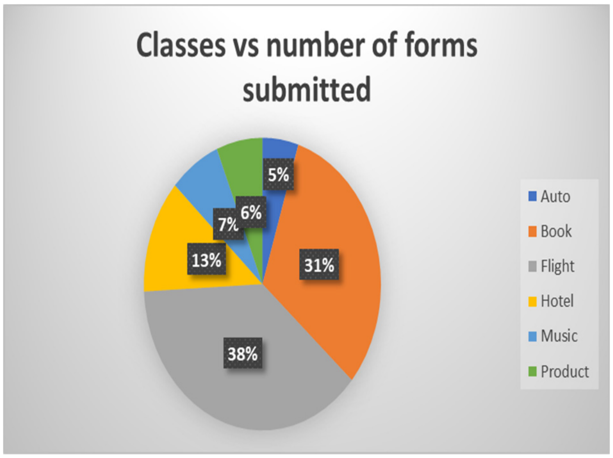

FIGURE 5. Percentage of correctly submitted forms submitted per class.

algorithms. Results have shown that SVM has performed better than Knn. Figure 5 shows the percentage of forms submitted per class. This percentage can vary with the number of URLs. The size of the database decreases as many of the URLs which do not fulfill the criteria of searchable forms are rejected. Figure 6 shows the comparisons for similarity detection using cosine similarity, Simhash and hybrid technique of Redis +Simhash. From the figure it evident that Redis+Simhash has given better results.

Table 4 show the performance of system in terms of classification of web forms that can not be submitted. Now the system has to submit searchable forms. For this confusion

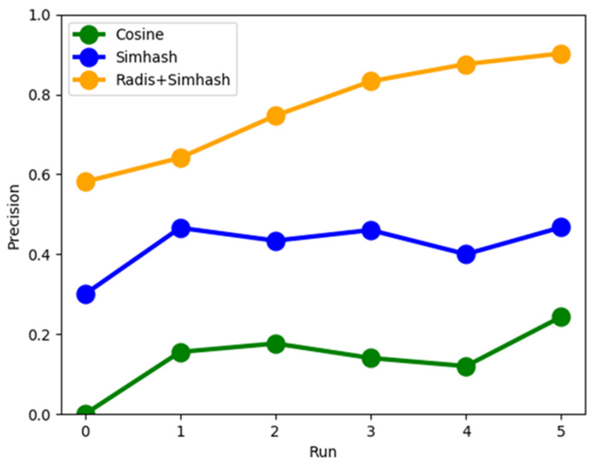

FIGURE 6. **Comparison of similarity detection of SIMHAR with cosine and Simhash.**

matrix is generated and classification performance is checked using SVM and KNN.

VI. DISCUSSION

A. MAIN CONTRIBUTION

After carefully review the literature, we found that focused hidden web crawling using distribution is still and unexplored area. The crawler named as SIMHAR is proposed and implemented based on this research gap. Results have proved that crawler detect and submit forms efficiently. Similarity detection in SIMHAR is hybrid technique based on Simhash and Redis server. Results are promising on compared with existing technologies. Following point discuss the other contributions

1. The values of the graph i.e classes vs number of forms submitted in figure 5, can vary with the number of available URLs because not all URLs would fall in the category of hidden web sites. Even if the dataset is huge, these values depend on the searchable forms.
2. This crawler is a successful implementation of focused crawling in the hidden web, then crawler is put on distribution mode.
3. The similarity is implemented in two ways, cosine similarity is first modified by adding information gain, then it is used for finding similar terms for top k queries.
4. This system is scalable as it can handle the size of growing web databases. And it is scalable in terms of processing and storage as well.
5. The crawler has inherent Redis security and fault tolerance. If one Redis server will stop responding the other will come into working as shown in figure [3]. Redis port can be made open for specific clients only. In our future work, we will work on this by implementing the crawler for client-based web harvesting.
6. The crawler is efficient in computing similarity with Redis and simhash combination.
7. The crawler work with both pre and post query approaches.
8. We have implemented stopping criteria's with which crawler resources will never deplete. As the number is fixed for forms as well as new found URLs.

VII. CONCLUSION

In this paper, we have proposed Redis based distributed web crawler for the hidden web called SIMHAR. Its resultant pages can be used for indexing and harvesting. The evaluation is detailed with a controlled environment using DMOZ, jasmine and amazon as the seed directories. We expect that system on being expanded to full scale can perform even better. The future work will introduce deep learning, and enhanced rejection criteria so that the goal of minimize visit and maximize the number of hidden websites could produce more promising results.

REFERENCES

[1] A. Heydon and M. Najork, "Mercator: A scalable, extensible Web crawler," *World Wide Web*, vol. 2, no. 4, pp. 219–229, Dec. 1999.
[2] G. Mohr, M. Stack, I. Rnitovic, D. Avery, and M. Kimpton, "An introduction to heritrix," in *Proc. 4th Int. Web Archiving Workshop*, 2004, pp. 109–115.
[3] M. Cafarella and D. Cutting, "Building nutch: Open source search," *Queue*, vol. 2, no. 2, pp. 54–61, 2004.
[4] M. Yadav and N. Goyal, "Comparison of open source crawlers—A review," *Int. J. Sci. Eng. Res.*, vol. 6, no. 9, pp. 1544–1551, 2015.
[5] M. Herrmann, K.-C. Ning, C. Diaz, and B. Preneel, "Description of the YaCy distributed Web search engine," KU Leuven ESAT/COSIC, IMinds, Tech. Rep., 2014.
[6] Y. Li, Y. Wang, and J. Du, "E-FFC: An enhanced form-focused crawler for domain-specific deep Web databases," *J. Intell. Inf. Syst.*, vol. 40, no. 1, pp. 159–184, Feb. 2013.
[7] L. Barbosa and J. Freire, "An adaptive crawler for locating hiddenwebentry points," in *Proc. 16th Int. Conf. World Wide Web WWW*, 2007, p. 441.
[8] L. Barbosa and J. Freire, "Searching for hidden-Web databases," in *Proc. WebDB*, vol. 5, 2005, pp. 1–6.
[9] B. Zhou, B. Xiao, Z. Lin, and C. Zhang, "A distributed vertical crawler using crawling-period based strategy," in *Proc. 2nd Int. Conf. Future Comput. Commun., ICFCC*, vol. 1, May 2010, pp. 306–311.
[10] W. Gao, H. C. Lee, and Y. Miao, "Geographically focused collaborative crawling," in *Proc. 15th Int. Conf. World Wide Web - WWW*, 2006, p. 287.
[11] J. Yu, M. Li, and D. Zhang, "A distributed Web crawler model based on cloud computing," in *Proc. 2nd Inf. Technol. Mechatronics Eng. Conf. (ITOEC)*, vol. 66, 2016, pp. 276–279.
[12] F. Ye, Z. Jing, Q. Huang, and C. Hu, "The research and implementation of a distributed crawler system based on Apache Flink," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, vol. 3, 2018, pp. 90–98.
[13] H. M. Moftah and S. M. Abuelenin, "Elastic Web crawler service-oriented architecture over cloud computing," *Arabian J. Sci. Eng.*, vol. 43, pp. 8111–8126, May 2018.
[14] D. Gunawan, Amalia, and A. Najwan, "Improving data collection on article clustering by using distributed focused crawler," *J. Comput. Appl. Informat.*, vol. 1, no. 1, pp. 39–50, 2017.
[15] H. T. Y. Achsan and W. C. Wibowo, "A fast distributed focused-Web crawling," *Procedia Eng.*, vol. 69, pp. 492–499, Jan. 2014.
[16] M. Boanjak, E. Oliveira, J. Martins, E. M. Rodrigues, and L. Sarmento, "TwitterEcho: A distributed focused crawler to support open research with Twitter data," in *Proc. 21st Int. Conf. Companion World Wide Web WWW Companion*, 2012, pp. 1233–1239.
[17] H. Xu, K. Li, and G. Fan, "An improved strategy of distributed network crawler based on Hadoop and P2P," in *Proc. Int. Conf. Appl. Techn. Cyber Secur. Intell.*, vol. 2, 2019, pp. 849–855.
[18] J. Madhavan, D. Ko, and A. Rasmussen, "Google ' s Deep-Web Crawl," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1241–1252, Aug. 2008.

[19] A. Bergholz and B. Childlovskii, "Crawling for domain-specific hidden Web resources," in *Proc. 7th Int. Conf. Properties Appl. Dielectr. Mater.*, Dec. 2003, pp. 125–133.
[20] J. Cope, N. Craswell, and D. Hawking, "Automated discovery of search interfaces on the Web BT," in *Proc. 14th Australas. Database Conf. (ADC)*, vol. 17, 2003, pp. 181–189.
[21] L. Barbosa and J. Freire, "Combining classifiers to identify online databases," in *Proc. 16th Int. Conf. World Wide Web WWW*, 2007, p. 431.
[22] A. Kashyap, V. Hristidis, M. Petropoulos, and S. Tavoulari, "Effective navigation of query results based on concept hierarchies," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 540–553, Apr. 2011.
[23] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, "Structured databases on the Web," *ACM SIGMOD Rec.*, vol. 33, no. 3, p. 61, Sep. 2004.
[24] J. Madhavan, A. Halevy, S. Cohen, X. L. Dong, S. R. Jeffery, and D. Ko, and C. Yu, "Structured data meets the Web: A few observations," *IEEE Data Eng. Bull*, vol. 29, no. 4, pp. 19–26, Dec. 2006.
[25] L. Barbosa and J. Freire, "Siphoning Hidden-Web Data through Keyword-Based Interfaces," *J. Inf. Data Manage.*, vol. 1, no. 1, pp. 133–144, May 2010.
[26] P. Zerfos, J. Cho, and A. Ntoulas, "Downloading textual hidden Web content through keyword queries," in *Proc. 5th ACM/IEEE-CS Joint Conf. Digit. Libraries (JCDL)*, 2005, pp. 100–109.
[27] J. Caverlee, L. Liu, and D. Buttler, "Probe, cluster, and discover: Focused extraction of QA-pagelets from the deep Web," in *Proc. 20th Int. Conf. Data Eng.*, Apr. 2004, pp. 103–115.
[28] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang, "DIADEM: Thousands of websites to a single database," *Proc. VLDB Endowment*, vol. 7, no. 14, pp. 1845–1856, Oct. 2014.
[29] P. Liakos, A. Ntoulas, A. Labrinidis, and A. Delis, "Focused crawling for the hidden Web," *World Wide Web*, vol. 19, no. 4, pp. 605–631, Jul. 2016.
[30] C. Sadowski and G. Levin, "SimHash?: Hash-based similarity detection," Google, Tech. Rep., 2007.
[31] G. Valkanas and A. Ntoulas, "Rank-aware crawling of hidden Web sites," in *Proc. WebDB*, 2011, pp. 1–6.
[32] S. Liddle, D. Embley, D. Scott, and S. H. Yau, "Extracting data behind Web forms," in *Proc. Int. Conf. Conceptual Modeling*. Berlin, Germany: Springer, 2002, pp. 402–413.
[33] P. G. Ipeirotis, L. Gravano, and M. Sahami, "Automatic classification of text databases through query probing," in *Proc. Int. Workshop World Wide Web Databases*, Berlin, Germany: Springer, 2000, pp. 245–255.
[34] C. Castillo, A. Nelli, and A. Panconesi, "A memory-efficient strategy for exploring the Web," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI Main Conf.)(WI)*, Dec. 2006, pp. 680–686.

SAWROOP KAUR received the M.Tech. degree from Lovely Professional University, Punjab, India, where she is currently pursuing the Ph.D. degree in computer science.

G. GEETHA (Member, IEEE) is currently a Professor in computer science and the Head of the Division of Research and Development, Lovely Professional University, Punjab. She works in the area of cybersecurity. She has published several research articles and has graduated six Ph.D. students.