Analysis of Semantic and Non-Semantic crawlers

Shridevi s, Shashwat Sanket, Jayraj Thakor, Dhivya M

Vellore Institute of Technology, Chennai, India

Abstract

A focused crawler goes through the world wide web and selects out those pages that are apropos to a predefined topic and neglects those pages that are not matter of interest. It collects the domain specific documents and is considered as one of the most important ways to gather information. However, centralized crawlers are not adequate to spider meaningful and relevant portions of the Web. A crawler which is scalable and which is good at load balancing can improve the overall performance. Therefore, with the size of web pages increasing over internet day by day, in order to download the pages efficiently in terms of time and increase the coverage of crawlers distributed web crawling is of prime importance. This paper describes about different semantic and non-semantic web crawler architectures: broadly classifying them into Non-semantic (Serial, Parallel and Distributed) and Semantic (Distributed and focused). An implementation of all the aforementioned types is done using the various libraries provided by Python 3, and a comparative analysis is done among them. The purpose of this paper is to outline how different processes can be run parallelly and on a distributed system and how all these interact with each other using shared variables and message passing algorithms.

Keywords

Semantic Crawler, Serial, Parallel, Distributed, message passing, shared variables

1. Introduction

A web crawler, also known as a spiderbot is a system made up of a program or an automated script that downloads web pages on a large scale. Web crawlers are used in various applications and in diverse domains. In fact, web crawling is one of the impact factors for the growth of internet in domains like marketing and E-commerce. In E- commerce, crawlers can be used for price comparison and to monitor the recent market trends. Similarly, it can be used to predict stock market movements by analysing social media content blogs and other data from different websites. Web crawlers are primary component of web search engines whose purpose is to collect web pages in bulk, index them and execute the user-defined query to find the web pages.

A similar use is web archiving where the web pages are collected and preserved or stored for future use. Along with the above mentioned uses web crawlers are also used to create a replica of visited pages which are processed by search engine for faster search optimization and web data mining to analyse statistically. Also, web crawlers are used to collect specific information like harvesting or collecting spam email addresses or application testing. Due to rapid increase of web pages and most of the data on web are unstructured, the semantic crawlers are used for retrieval of context relevant web pages. Semantic crawlers have different architectures like distributed, parallel, focused and increment crawler.

Today, web crawlers form an important part of various software services to evolve into large scale integrated distributed software proving that they are not just a program preserving a list of pages to be crawled. The web crawler is the principal and time demanding element of web search engine. It consumes huge amount of CPU time, memory and storage space to crawl through ever increasing and dynamic web. The time it consumes to crawl through web should be as small as possible to maintain its

ISIC'21: International Semantic Intelligence Conference, February 25-27, 2021, Delhi, India
EMAIL: shridevi.s@vit.ac.in (S. Shridevi); dhivya.m2019@vitstudent.ac.in (M. Dhivya).
ORCID: 0000-0002-6927-1998 (M. Dhivya

recent updates of the search outputs. Parallel and distributed processing is one way to increase the speed of crawling process due to technological advancement and improvement in hardware architectures.The work consists of implementation and comparison between different web crawler architecture namely Serial, Parallel and Distributed. The purpose of this work is to outline how we can increase the processing capabilities of web crawlers and get the query output in lesser amount of time. This paper covers detailed information about how different processes can be executed on parallel and on a distributed system and how all these interact with each other using shared variables and message passing algorithms.

2. Existing Work and Literature Survey

In this section, the recent works related to crawler processing is described. In "Speeding up the web crawling process on a multi-core processor using virtualization" [1] by Hussein Al- Bahadili, Hamzah Qtishat, and Reyadh S. Naoum, they have presented and analysed their new approach to increase the crawler efficiency in terms of time through virtualization using multi- core processor. In their work they have divided the multi-core processor into many VMs (Virtual Machines), so that the task can be executed concurrently on different data. In addition to this they have also described their implementation and analysis of VM-based distributed web crawler after rigorous testing.

J. Cho, Hector G., L. Page [2] in their work have described in what sequence or in what order the URLs must be visited by the crawler to obtain the important pages first. This method of obtaining pages of prime importance rapidly, helps to save time when a crawler is unable to go through the increasing and dynamically changing web. In this work they created a dataset by downloading an image of Stanford Webpages and performed experiment by modifying and using different large-scale and small-scale crawlers like PageRank Crawler, Breadth-first and Depth-first search crawler and Backlink-based crawlers.

"Google"s Deep-Web Crawl" by J. Madhavan,
D. Ko et al [3] is another notable work describing how to crawl the contents of deep-web which is used in Google search engine. They have described a system to extract deep-web content which includes pre-computing submissions for each HTML form and adding the resulting HTML pages into a search engine index. The entire system is based on achieving three main goals. The first goal is to develop an approach that is time saving, automatic and scalable to index the hidden web content from HTML forms that are varied in domains and are in languages from all over the world. The second aim is to develop two types of algorithm, one that can identify the inputs that accepts only specific value types and other to accept a keyword to select input values for text search inputs. The third aim is to develop an algorithm that goes through the possible input combinations to identify and generate URLs suitable for web search index.

Anirban Kundu, Ruma Dutta, Rana Dattagupta, and Debajyoti Mukhopadhyay in their paper "Mining the web with hierarchical crawlers – a resource sharing based crawling approach" [4] have proposed an extended web crawling method to crawl over the internet on behalf of search engine. The approach is combination of parallelism and focused crawling using multiple crawlers. The algorithm divides the entire structure of the website into many levels based on hyperlink structure to download web pages from the website and the number of crawlers is dynamic at each level. The number of crawlers required is determined based on the demand at run time by and by developing a thread-based program using the number of hyperlinks from the specificpage.

M. Sunil Kumar and P. Neelima in their work "Design and Implementation of Scalable, Fully Distributed Web Crawler for a Web Search Engine" [5] have presented Dcrawler which is highly scalable and distributed. The core features of the presented crawler are decentralization of tasks, an assignment function that partitions the domain for the crawler to crawl effectively, cooperative ability in order to work with other web servers and platform independence. For assignment function Identifier-Seeded Consistent Hashing have been used. On performing tests using distributed crawlers they concluded that the Dcrawler performs better than other traditional centralized crawlers and also performance can be improved with addition of more crawlers.

T. Patidar and A. Ambasth in their paper "Improvised Architecture for Distributed Web Crawling" [6] have proposed reliable and efficient methods for a web crawler that is scalable. In addition to this they have

discussed challenges and issues regarding web structure, job scheduling, spider traps and URL canonicalization. The components of their proposed work include Child Manager, Cluster Manager, Bot Manager and incremental batch analyser for re-crawling. Their results show that they have successfully implemented distributed crawler along with politeness techniques and selection policies but still they face challenges like resource utilization.

The work "A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration" [7] by E. Dragut et al. describes an algorithm which extracts and maps query interfaces into a hierarchical representation. The algorithm is divided into 4 steps namely Token Extraction, Tree of Fields, Tree of Text Tokens and Integration and therefore they convert extraction algorithm into integration algorithm. They carried out experiments on three different datasets (ICQ, Tel8 and WISE) and evaluated the algorithm based on performance metrics like leaf labelling, Schema Tree Structure and Gold Standard.

D. H. P. Chau, S. Pandit, S. Wang, and C. Faloutsos have described parallel crawling by illustrating it on an online auction website in their work "Parallel Crawling for Online Social Networks" [8]. They have presented this work for online social networks. They have dynamic assignment architecture which ensures that failing of one crawler does not affect another crawler and that there is no redundant crawling. They visited about 11 million users out of which approximately 66,000 were completely crawled. J. Cho and H. Garcia-Molina [9] proposed different architectures for parallel web crawlers, metrics to evaluate the performance of parallel web crawlers and the issues related to parallel crawling. They described issues like Overlap, quality and communication bandwidth and advantages of parallel crawlinglike scalability, Network-load dispersion and Network- load reduction.

C. C. Aggarwal, F. Al-Garawi, and P. S. Yu in their work "Intelligent crawling on the world wide web with arbitrary predicates" [10] have described intelligent crawling as a method that learns properties and features of the linkage structure of WWW while crawling. The technique proposed by them is more generalized than focused crawling which is based on pre-defined structure of web. The intelligent crawling described by them is applicable to web pages that support arbitrary user-defined topical and keyword queries. The technique described is capable of reusing the information gained from previous crawl in orderto crawl more efficiently the next time.

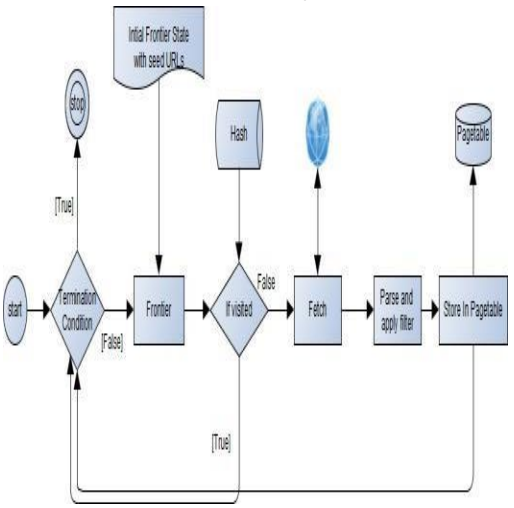

Figure 1 Flow of Serial Web Crawler

3. Architectures and Implementation

3.1 Serial Web Crawler

The crawler maintains a list of unvisited URLscalled the frontier which acts a queue. The list is initialized with seed URLs. Each crawling loop involves picking the next URL from the frontier to crawl, checking if the URL is previously visited or not, if not visited then fetch the page corresponding to the URL through HTTP, followed by parsing the retrieved page to extract the URLs and application specific information, and finally adding the unvisited URLs to the frontier. The crawling process may be terminated when a certain number of pages have been crawled. If the crawler is ready to crawl another page and the frontier is empty, the situation signals a dead-end for the crawler. The crawler has no new page to fetch and hence it stops. Figure 1 shows the flow of a basic serial web crawler. The complete implementation of the above model can be found in Implementation 1.

Algorithm:

1. Initialise a constructor crawler with variable pageTable and revPageTable assigns to HashMap
2. define a method get_seed() to get the the seed_url

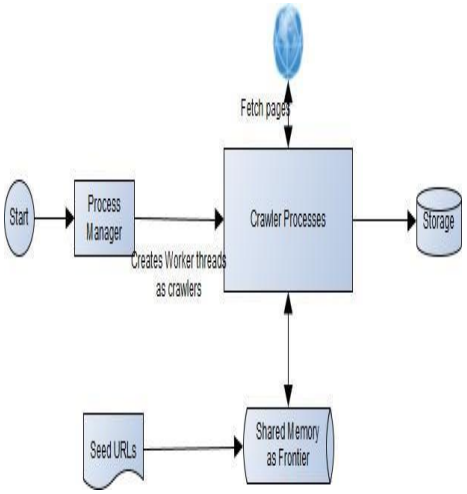

Figure 2 Flow of parallel webcrawler

3. Parse the seed_url and save in hostname
4. Initialize the frontier.
5. Define a method test seed url
6. if the url has same hostname as that of seed url then return false otherwise return true.
7. Define a method to get all urls
8. try fetching the url and throw exception
9. parse the page using HTML parser
10. for each link in parsed page find all „a"
11. if parsed link is not safeURL
12. throw invalid URL
13. otherwise append url and link
14. //end if
15. //end for each
16. Define a mothod crawl
17. for each current url in frontier
18. fetch all the url and update the frontier length.
19. for each url in urls
20. if current url is in page table then push url
21. if url is not in frontier and test_seed_url(url) then push url to frontier.
22. //end if
    1. //end if
    2. //end for each
    3. //end for each

3.2 Parallel Web Crawler

Parallel crawlers can be understood as several modified serial crawlers running as separate processes. These multiple processes run in parallel thus named parallel web crawler. Figure 2 shows the flow chart of the working of the parallel web crawler. Here we created a process pool that is managed by the process manager, which is also responsible for spawning and scheduling new processes. And a shared memory that is used as Frontier. Note our crawler is a simple parallel web crawler. Although there are many different ways of URL partitioning as mentioned in [11]. But the main aim here is to create a Baselinemodel.

Algorithm: Crawler

1. Initialise a class crawler with a constructor.
2. Define method testseedurl with seed url as parameter
3. if hostname is same as that of seed url then return false
4. otherwise return true
5. Define method getallurl with url as parameter
6. fetch and parse the html page
7. for each parsedPage.findAll(„a",href=TRUE)
8. if (!safeURL(link)) then throw URL invalid
9. otherwise append url urls.append(url+link)
10. Initialise the daemon server.

Algorithm: Frontier

1. Initialise a class frontier_manager witha constructor.
2. Initialise the process pool and seed_url.
3. Initialise the frontier with the global list and assign token to each process bylock acquire and release method
4. for each url in urls
5. if url is not in frontier push url into frontier and self-release the lock
6. //end if
7. Define a method to write to the url to index table
8. The index acquires the lock and check if seed url is not equal to key in index table then add url to table.
9. otherwise indextable[local_seed_url].extend(url)
10. Define a method Make_write and writethe urls to frontier and index table
11. define a static method crawl and initialise a crawler

12. return the seed url
13. Define a method start and create apool process
14. close the pool process

The above Baseline code uses multiprocessing for creating and managing multiple processes. Here lock-based system is used to access the shared memory space. The crawler code is very similar to that of serial web crawler the modification is done for the Frontier. The Frontier spawns two crawler workers to fetch the pages.

3.3 Distributed Web Crawler

Distributed web crawlers [14,16,17] are a technique in which many computers participate by providing their computing bandwidth in the crawling process. The proposed architecture acts as a baseline for this technique. In this, there is a central server as nameserver and four other servers as workers as crawlers. Here we used the dynamic assignment as a policy where the nameserver dynamically assigns the URLs and balances the load. Note the nameserver here is not responsible for crawling in order to reduce theworkload.

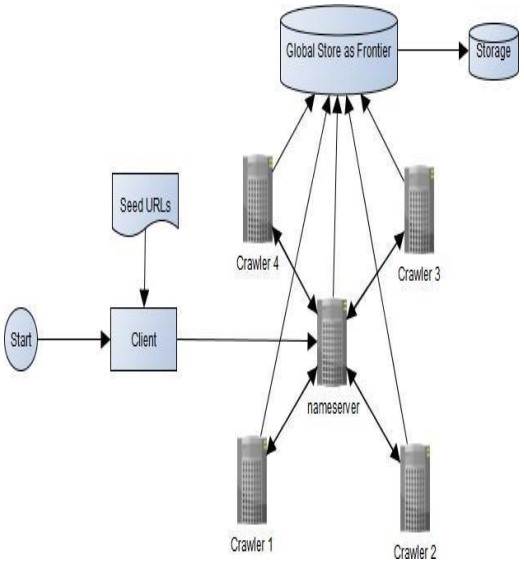

Figure 3 flow of distributed web crawler using client- server architecture.

Apart from the dynamic assignment job of the nameserver, it is also responsible for monitoring the hearth beat and other meta information of these worker crawlers. Since the nameserver can be a single point of failure (SPOF) during the task. To avoid this, the nameserver saves all the meta- information in the form of checkpoints in the global store. On failure of a nameserver, one ofthe crawlers will be elected as the nameserver, and the new nameserver will fetch the latest checkpoint and continue the task. Note here the crawlers are also receiving the heartbeat signal of a nameserver, in order to identify when the nameserver is down.

Before going the flow, here we used two frontiers as local frontier and global frontier. local frontier is the frontier of the worker instance where as a global frontier is part of the global store .The client will trigger the nameserver by providing the seed URLs to crawl, the nameserver will initialize the global frontier with the seed URLs, and will dynamically assign the URLs to the respective the crawlers local frontier. The crawlers individually be acting as serial web-crawler with its own DNS resolver, Frontier queue, and pagetable. For filtering the URLs they will also communicate with the global store to check if visited or not. Upon completing the crawling process, thecrawler will dump the pagetable in the common storage and will ask the nameserver to reallocate the new seed URLs. This process continues till termination triggered by the nameserver.

Algorithm: Crawler

1. Initialise a class crawler with a constructor.
2. Define method testseedurl with seed url as parameter
3. if hostname is same as that of seed url then return false
4. otherwise return true
5. Define method getallurl with url as parameter
6. fetch and parse the html page
7. for each parsedPage.findAll(„a“,href=TRUE)
8. if (!safeURL(link)) then throw URL invalid
9. otherwise append url urls.append(url+link)
10. Initialise the daemon server.

Algorithm:

1. Initialize class Frontier_Manager() and variable seed url.
2. create a index table using HashMap function
3. Define method testseedurl with seed url as parameter
4. if hostname is same as that of seed url then return false

5. otherwise return true
6. Define method write_to_frontier with seed_url and host urls a parameter.
7. for each url in urls
8. if(!(url in frontier and test_seed_url(url))) then self.frontier.append(url.strip())
9. Define method write_to_index_table.
10. if local seed url is not present in index table then add the local seed url
11. otherwise index_Tble(local_seed_url). extend(urls)
12. write to frontier and index_table the local_seed_url and urls.
13. fetch the method crawl by using index and crawler variable
14. end if
15. end each for.

3.4. Semantic Distributed web crawler

Distributed Semantic web crawlers are used for crawling both semantic web pages in RDF/OWL format and HTML pages. The distributed semantic crawler uses a component called page analyser for understanding the page context. The ontology analyser creates models for fetched OWL/RDF pages and these models are stored. Later these models are matched with the stored Ontology to make crawling decision

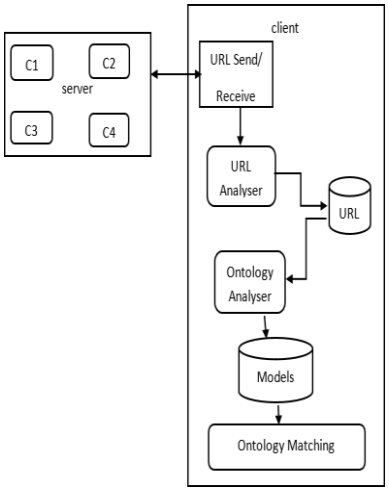

Figure 4: Architecture of Distributed Semantic web crawler

Algorithm: Crawler controller

1. Initialise string with seed URL
2. check the whether the string is present in Database
3. check if (seed URL exist) print already exist
4. otherwise insert the URL details
5. assign variable for statement and add the seed details to the database
6. if (statement! = empty) {execute statement}
7. otherwise print statement not executed.

Algorithm: Model Extraction

1. Initialise variable id, url, html, langType
2. create an object and read the url.
3. Repeat till the statement is present
4. define variables and get the subject, predicate, object and URI.
5. if object is URIResource then get URI and asiign to ob.
6. //end if
7. if langtype is HTML then
8. if subject does not contain # and is not null then save subject to database,
9. if predicate does not contain # and is not null then save object to database.

3.5 Focused web crawlers

Focused web crawlers [18] are to use to collect web pages on a specific topic. These crawlers search the entire web on a predefined topic which in turn avoids irrelevant information to the user. Focused crawler saves the computational resource.

Semantic focused crawler has multi thread and each thread takes a web page with highest dynamic semantic relevance from priority queue. The main work of the thread is to parse the various hyperlink and add them to the priority queue. Thus, the priority queue has the details of the web page that has to parsed by the thread Semantic focused crawler has another temporary queue which maintains the visited web

pages. The thread also checks this temporary queue for visited web pages.

Algorithm: semantic focused crawler
Q: Priority Queue
DSR: Dynamic Semantic relevance
Link: Queue of traversed URL

1. Initialize priority queue Q with seed URLs
2. Repeat till (!Q.empty() ||fetch cnt 6 Limit) {
3. web page.url = Q.top.getUrl (); //Get most relevant single URL from priority queue
4. Fetch and parse web page.url;
5. web page.urls = extract URLs (hyperlinks) from web page.url; //List of URLs
6. For each web page.urls {
7. already exist = Check web page.urls[i] in Links; //Check for duplicates
8. If (!already exist) {
9. Enqueue web page.urls[i] in Links;
10. Fetch and parse web page.urls[i];
11. Compute DSR of web page.urls[i];
12. Enqueue (web page.urls[i], DSR) in Q;
13. Store (web page.urls[i], DSR) in local database;
14. } //end of If
15. } //end of For each
16. } //

Here using Pyro4 python library to stimulate the described architecture. Beautiful Soup to parse the HTML pages.

4. Results

Fig 6 shows the testing of all the semantic and non- semantic crawlers for a given website. From the table in figure 6 the total number of test cases are 30 out of which the non-semantic crawlers (distributed crawler outperforms in 20 cases; parallel crawler outperforms in 8 and serial crawler outperforms in only 2 cases) and semantic crawlers (semantic distributed crawler outperforms in 24 cases and focused crawler in 26 cases). Therefore, distributed crawler achieves an accuracy of about 66.67%, parallel crawler achieves an accuracy of 26.67 %, serial crawler gives an accuracy of about 0.67%, semantic distributed crawler gives an accuracy of 80% and focused crawler gives an accuracy of about 86.66%.

Sl. No	websites	No. of test cases	Distributed Crawler	Parallel Crawler	Sequential Crawler	Semantic Distributed Crawler	Focused crawler
1	vit.ac.in	7	4	2	1	5	6
2	chennai.vit.ac.in	11	7	3	1	8	10
3	en.wikipedia.org	5	3	2	0	4	5
4	bbc.co.uk	4	3	1	0	4	3
5	academia.org	3	3	0	0	3	2

Table 1 Number of times a specified crawler outperforms other crawlers

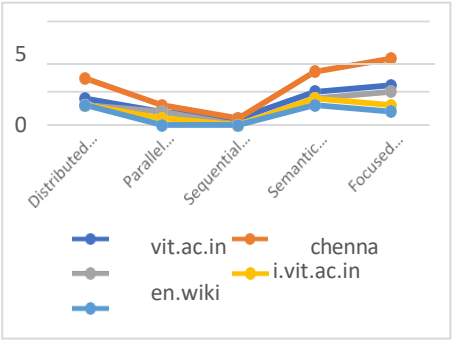

Figure 6 Graphical representation of number of times a specified crawler outperforms

Fig 6 shows the graphical presentation of the number of times a specified crawler outperforms.

5. Conclusion

It can be concluded that for majority of time, a focused crawler and semantic distributed crawler gives the best result for crawling a specific website. From the result it is also clear that focused crawler works well as the number of crawling increases.

References

[1] Al-Bahadili, H. & Qtishat, Hamzah & Naoum, Reyadh. (2013). Speeding Up the Web Crawling Process on a Multi-Core Processor Using Virtualization. International Journal on Web Service Computing. 4. 19-37. 10.5121/ijwsc.2013.4102.

[2] Junghoo Cho, Hector Garcia-Molina, and

Lawrence Page. 1998. Efficient crawling through URL ordering. In Proceedings of the seventh international conference on World Wide Web 7 (WWW7). Elsevier Science Publishers B. V., NLD, 161–172.

[3] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. 2008. Google‟s Deep Web crawl. Proc. VLDB Endow. 1, 2 (August 2008), 1241–1252.

[4] Kundu, Anirban & Dutta, Ruma & Dattagupta, Rana & Mukhopadhyay, Debajyoti. (2009). Mining the web with hierarchical crawlers - A resource sharing based crawling approach. IJIIDS. 3. 90-106. 10.1504/IJIIDS.2009.023040.

[5] Kumar, M. & P, Neelima. (2011). "Design and Implementation of Scalable, Fully Distributed Web Crawler for a Web Search Engine". International Journal of Computer Applications. 15. 10.5120/1963-2629.

[6] Patidar, T., & Ambasth, A. (2016). Improvised Architecture for Distributed Web Crawling. International Journal of Computer Applications, 151, 14-20.

[7] Kabisch, Thomas & Dragut, Eduard & Yu, Clement & Leser, Ulf. (2009). A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration.PVLDB.325-336. 10.14778/1687627.1687665.

[8] Duen Horng Chau, Shashank Pandit, Samuel Wang, and Christos Faloutsos. 2007. Parallel crawling for online social networks. In Proceedings of the 16th international conference on World Wide Web (WWW „07). Association for Computing Machinery, New York, NY, USA, 1283–1284.

[9] Cho, Junghoo, and Hector Garcia-Molina. "Parallel crawlers." Proceedings of the 11th international conference on World Wide Web. 2002.

[10] Aggarwal, Charu & Al-Garawi, Fatima & Yu, Philip. (2001). Intelligent Crawling on the World Wide Web with Arbitrary Predicates. 96-105. 10.1145/371920.371955.

[11] Parallel Crawlers Junghoo Cho, Hector Garcia-Molina University of California, Los Angel

[12] Naresh Kumar, Manjeet Singh (2015). Framework for Distributed Semantic Web Crawler. IEEE - International Conference on Computational Intelligence and Communication Networks.

[13] K.Lokeshwaran, A.Rajesh (2018). A Study of Various Semantic Web Crawlers and Semantic Web Mining. International Journal of Pure and Applied Mathematics Volume 120 No. 5 2018, 1163-1173.

[14] F. Liu and W. Xin, 2020 "Implementation of Distributed Crawler System Based on Spark for Massive Data Mining," 2020 5th International Conference on Computer and Communication Systems (ICCCS), Shanghai, China, 2020, pp. 482-485, doi: 10.1109/ICCCS49078.2020.9118442.

[15] Rajiv, S and Navaneethan, C, 2020, "Keyword Weight Optimization using Gradient Strategies in Event Focused Web Crawling" Pattern Recognition Letters 01678655 CrossRef.

[16] S. K. Bal and G. Geetha, "Smart distributed web crawler," 2016 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, 2016, pp. 1-5, doi: 10.1109/ICICES.2016.7518893.

[17] Wang, HongRu, et al.2018 "Anti-Crawler strategy and distributed crawler based on Hadoop." *2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA)*. IEEE.

[18] Boukadi, K., Rekik, M., Rekik, M., & Ben-Abdallah, H. (2018). FC4CD: a new SOA-based Focused Crawler for Cloud service Discovery. *Computing*, *100*(10), 1081-1107.