International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)
Impact Factor: 5.22 (SJIF-2017),e-ISSN:2455-2585
International Conference on Smart Cities (ICSC-2019)
Volume-05, Special Issue_March_2019.

Clickstream Based Parallel Focused Web Crawler

Naresh Kumar[1], Sarneet Singh Chitkara[2], Prashant Solanki[3]

Department of Computer Science & Engineering, Maharaja Surajmal Institute of Technology, New Delhi, India

Abstract. Due to the increasing size of the Web each day, there is always an ever-increasing need for Web crawlers and searching algorithms that provide efficient scalable crawling and searching mechanisms for indexing the Web. Due to the voluminous size of the Web, single process crawlers are no longer enough and linkbased metrics, no longer produce a properly ordered answered set. Traditional linkbased metrics in a parallel crawling environment come with a lot of communication overhead, which reduces the speed of the crawler and consumes unnecessary bandwidth. The aim of this paper is to propose an architecture for parallel crawler that relies on content-based metrics for determining page importance. The page importance is calculated using different weights assigned to each metric collected using the clickstream based on their relevance to the context. The experiment was done on a blog keeping in mind, the politeness factor in the crawling process to ensure that the process did not affect the normal functioning of the website.

Keywords: Crawler· Data acquisition· Indexing· Parallel processing· Search engines· Web mining.

1 Introduction

The World Wide Web has seen a multidimensional increase in content and volume. Search Engines are facing continuous challenges in indexing all these Web Pages as fast and as efficiently as possible. Web crawlers or Web spiders are automated programs/bots that browse through the World Wide Web in a systematic way with the purpose of Web indexing. A single process single instance crawler is no longer sufficient for crawling this huge amount [1]. There is a need of a Multi-Threaded Web crawler. This can be scaled further by running several crawler instances at once thus, further increasing the efficiency and performance of the crawler. Even if the issue of scalability is resolved, most crawlers are hindered by their ordering algorithms which may fail to provide a proper and precise ordered answer set to a query. An architecture for a parallel and focused crawler is designed for implementation of an efficient web crawler which comprises of several independent crawling agents that scrape a given Web site and communicate with each other in order to avoid crawling of same the URLs again and again. For ordering the crawled URLs in a meaningful way, clickstream analysis is done on each Web page to assign an Importance Value to each page which helps in filtering out less important pages and identifying more relevant pages. Clickstream analysis [2],[3],[4] helps in designing link-independent metrics which help in avoiding the costs associated with maintenance of sparse link matrix of Web graph. This is achieved using a script written in JavaScript, which stores a repository of various metrics like visits and downloads associated with a Web page. Along with this, Okapi BM25 [5] ranking function is used to determine context similarity while querying. Based on all these metrics, an ordering rule is designed which determines the Importance Value associated with each Web page and helps in ordering the URLs of a Web site.

2 Literature Survey

Extensive literature survey was conducted, and the following knowledge was obtained from the conducted survey.

- A prototype of a focused Web crawler which prioritizes the Web pages for effective download was proposed in [6]. It overcomes the problem of accurately predicting the topical focus of a page and prioritizing it without downloading the actual contents of the page. An algorithm was proposed to determine the relevance of the page. Furthermore, an algorithm is proposed in [7] to determine the priority of the page using a T-Graph structure which takes into consideration the various HTML elements of the source page, along with the Dewey Decimal System Classification as a basis to classify text into respective topical boundaries, to detect the topical focus of the unvisited link. The prototype implementation of Treasured-Crawler outperformed the context-based Web crawler in terms of accuracy with 14% improvement on crawls with generic seeds and 22% improvement on crawls with on-topic seed.
- A link-independent Web page importance metric for a parallel focused Web crawler was proposed in [2]. The importance value of a Web page calculated using a link-dependent importance metric relies on the downloaded index and the pages which have been already been crawled. Thus, the calculated rank of a page is different from its actual rank as a portion of the Web is still not known to the crawler. Detection of authoritative Web pages on the 'dark Web' also poses a challenge for link-based metrics. In [8] an attempt to overcome the drawbacks associated with link based metrics, the authors proposed a clickstream based importance metric to calculate the rank of different pages for effective crawling using a parallel focused crawler. Clickstream analysis is the process of collecting and analyzing the data about Web pages which various users visit in what order which is the result of the successive mouse clicks a user makes. A crawler based on this link-independent metric is more efficient as the need for communication between the

individual agents in a parallel crawler is completely eliminated. In [9] the architecture was proposed for a multi-agent Web crawler to implement the clickstream-based link-independent importance metric using the textual log files of the Web page. Server Logs from the author's college website were used for clickstream data. Five ordering rules were suggested to prioritize the Search Result Set which stores the list of Web pages ordered by the calculated importance values. An experiment was carried out using this crawler on the UTM website with 1721 indexed Web pages which resulted in a Final Ordered List consisting of 1061 distinct Web pages and out of the 660 dark Web pages on the website, 26 became visible.

- A cloud-based approach was proposed in [3], wherein the authors proposed a distributed crawler that runs on the cloud instances/machines. It involved an architecture consisting of an agent that perform crawling and a service to synchronize between the agents. It was using link and content based prioritization for relevance prediction and prioritization of Web pages. Agents were performing the worker role whereas synchronizer was responsible for orchestration between the agents and managing other aspects of the crawler. The major challenge faced was the problem of scalability. Use of NoSQL database enabled faster searching and an efficient storage of multitudes of data. As a result, the response time for 18223 queries was reduced to 40ms from 60ms earlier when running 16 crawler agents. Dataset used was a query that transacts 18223 transactions for speed measurement.
- In a service running on the cloud, there arises a need of load balancing. Load Balancing is concerned with which node will service a request so as to keep the load balanced in a cluster. Consistent Hashing as a load balancing scheme was proposed in [10],[11]. Consistent Hashing as its partitioning policy. This policy is claimed to be independent of the number of documents on servers and also can balance the load between crawling when they dynamically join or leave the system. Mathematically, the algorithm used power law to generate hash keys. Similar hashed indexes were crawled by the same crawler agent. The authors designed a simulator to simulate the crawling process and the results obtained were consistent with the proposition.
- A geo-distributed crawler, UniCrawl was proposed in [12]. UniCrawl orchestrated several geographically distributed sites, where each site operated an independent crawler and relied on already established techniques for fetching and parsing of the Web page content. The spread of domain space across various sites split the storage and computing resources while minimizing the inter-site communication cost.
- Implementation of a hierarchical multi-agent Web crawler for distributed crawling was demonstrated in [13], [14]. An application was developed to extract relevant data and keywords from the popular social networking website, Twitter. The Web spider initiated crawling through a predefined list of URLs (seeds) and downloaded the content of that Web page. The meta data and relevant content were then parsed and stored in a database. To improve the efficiency of downloading a large volume of data from the Web, the crawler was run on a distributed system (increased bandwidth) and the data obtained from the various Web pages was indexing by decentralized search infrastructures and archived using a permanent storage infrastructure [15]. The Twitter Search API was used to query against a real-time index of recent Tweets. One of the challenges was to make the application scalable and balanced and so a number of individual agents were designed to crawl in separate segments on the Web. Communication with the database was a crucial factor in determining the number of parallel crawlers and the most favorable result was produced using ADO.NET database by deploying 30-40 concurrent crawlers. A number higher than this range was unable to process all the requests efficiently and gave poor results

3 Problem Statement

Upon consulting, extensive literature on Web crawler, Clickstream metrics and Link based metrics the following problems were identified for a parallel crawler: -

- Link based metrics are not an adequate solution as they have a huge overhead associated with them and can be spoofed.
- Server Log based clickstream analysis is not a viable option as it is a threat to security. The server logs contain important information related to the server which can pose a serious threat to security. Also, parsing these server logs would require a lot of time and also storage space in the order of GBs.
- Scalability is a big issue in parallel crawlers. Having more instances doesn't simply mean more crawls per minute. The architecture has to be designed and tested thoroughly to find the best seed splits and to avoid overlapped crawls.
- Other than clickstream metrics, there also must be a factor that provides an answer set according to contextual similarity with the query. Thus, the architecture must be update to process the web pages on which clickstream metrics are being collected, for contextual data and that too without causing a huge impact on page load performance.

4 Proposed Crawler Architecture

In a parallel crawler, each crawler is multithreaded and direct inter-crawler communication is not possible. An orchestrator service is required to perform the critical tasks of seed partitioning and deduplication. The architecture for each parallel crawler agent and the complete architectures proposed are described below.

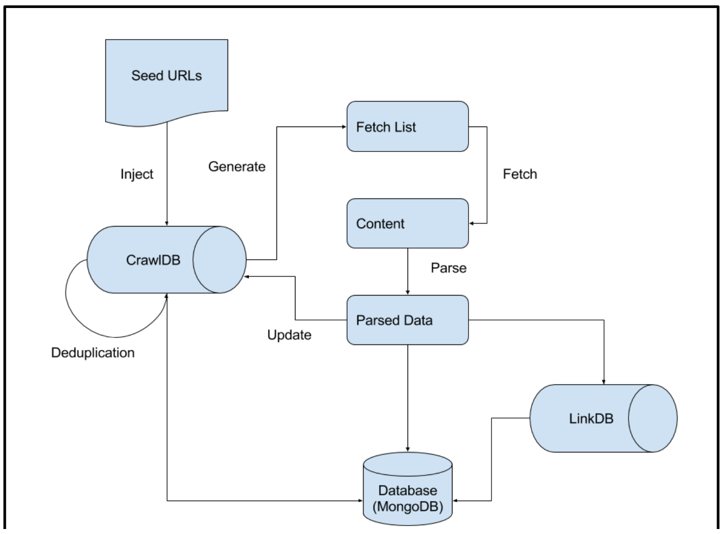


Figure 1: Each parallel crawler agent internal architecture

Individual Crawler internal architecture

The seed URLs after splitting are injected into the CrawlDB which maintains the information of all the known URLs. Based on the injected URLs, a fetch list is generated in which all URLs are to be fetched. The fetcher fetches the URLs in fetch list thus downloading the URL content and saving them in segments. The parser parses that downloaded content of each web page. The updates are also done by the fetcher for already crawled URLs. The pages. This data is saved onto themain Database and indexed for faster access.

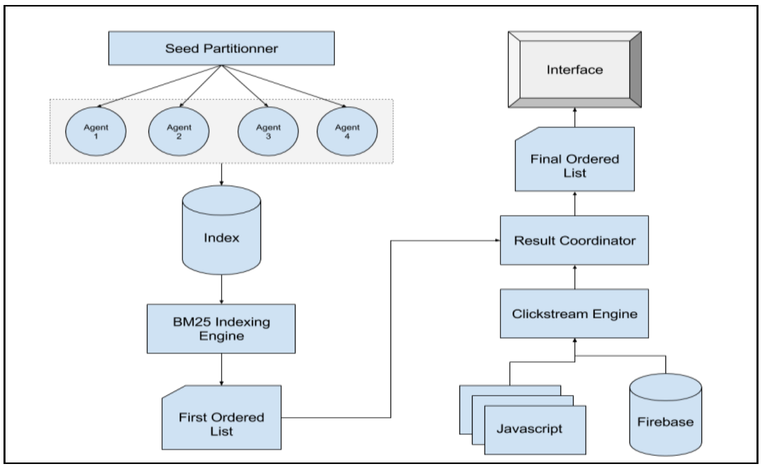


Figure 2: Architecture of clickstream based parallel web crawler

4.1Proposed Architecture for Clickstream based parallel Web Crawler

The seed partitioner partitions or distributes the seeds using a suitable partitioning policy, between the crawler agents. The seeds are distributed so as to avoid overlap and distribute the load evenly to improve efficiency. The crawlers crawl the seeds and all the results are de duplicated and Indexed. The BM25 ranking engine ranks these crawled and indexed pages according to their contextual similarity and assigns a score in the range 0 to 25 which is scaled to 0 to 100 for ease of calculation. The first ordered list contains the URLs ordered on the basis of their BM25 scores.

The Clickstream Engine comprises of the JavaScript which runs on the Web Pages and collects Clickstream metrics. All these metrics are stored in the Firebase real-time database for improved accuracy. The clickstream metrics and the First ordered list are used to calculate Importance value(IV) of each page. The pages are then ordered according to their IVs to generate the final ordered list.

4.1 Algorithm for calculation of IVs

Assumption 1: Pages with high ASCII sum are more important than those with lower ASCII sum

Assumption 2: Pages that were opened for longer duration are more important than others.

Assumption 3: Pages with higher no of visits are more important.

Algorithm - Importance Value Calculator **Input**: URL.
Output: Importance Value of URL.

Step 1: Start.

Step 2: Read URL.
Step 3: Download Web page content.
Step 4: Apply BM25 Ranking Function on the content for context similarity.
Step 5: Fetch clickstream analysis results on the URL to get visits (V), duration (D), downloads (Do).
Step 6: Calculate raw score for each URL using the formula:
Score = V + 1.5*BM + 0.75*log(D*A) + 5*Do.
Step 7: Normalize raw scores to get importance value.
Step 8: End.

Formula for calculating the Importance Values of the Web Pages

$$\textbf{Score} = V + 1.5*BM + 0.75*\log(D*A) + 5*Do$$

Here,

V	:	Total no. of visits on the page
BM	:	BM Rank of the page between 0-100
D	:	Document Length
A	:	ASCII sum value of the Web Page.
Do	:	No. of Download links in the Web Page.

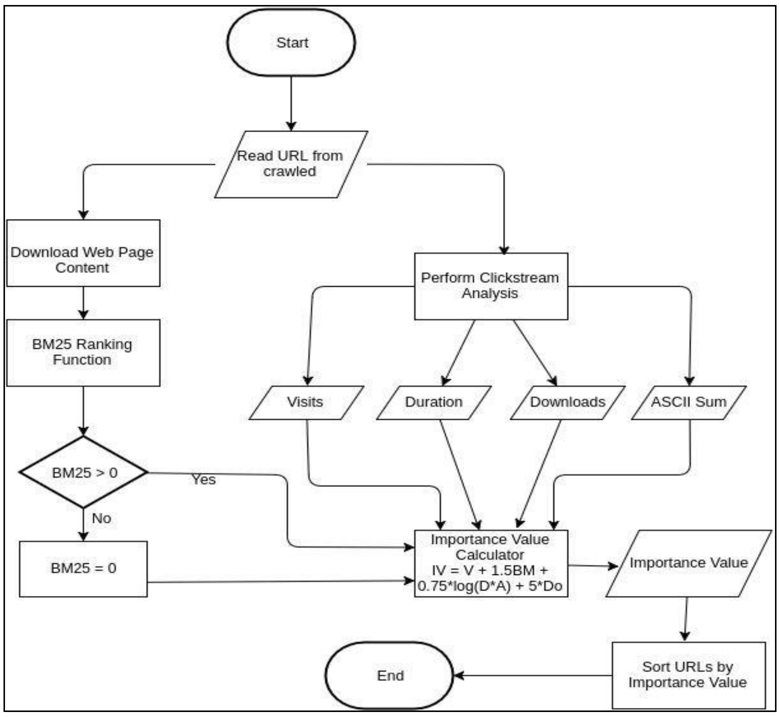

Start
Read URL from crawled
Download Web Page Content
Perform Clickstream Analysis
BM25 Ranking Function
Visits
Duration
Downloads
ASCII Sum
BM25 > 0
Yes
No
BM25 = 0
Importance Value Calculator IV = V + 1.5BM + 0.75*log(D*A) + 5*Do
Importance Value
Sort URLs by Importance Value
End

Figure 3: Flowchart depicting calculation of Importance Vectors

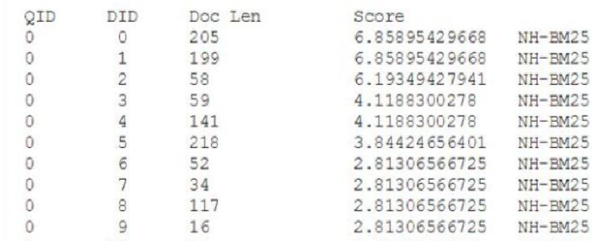
5 Experimental Setup and architectural requirements

- The architecture was implemented and deployed on Centos 7 Linux OS (Minimal edition)
- Hardware specs - Intel Core i5 6300HQ Quad Core CPU, 8GB RAM and SSD
- Service/Program requirements – Apache Nutch 2.3, MongoDB 3.2, ant, Java 8, Python 3.5.2 ● Dependencies – matplotlib, os, json, numpy, firebase Python packages.

6 Experiment

The proposed architecture was tested several times for different URLs. For experimental testing the results are taken from a startup website https://blog.kakcho.com. The JavaScript was globally deployed onto the websites Nov on at 13 10:20th November AM. The for-collection clickstream of data the Clickstream was collected data. and the stored crawling into process began on 15 firebase during the period. The real-time database ensured always up to date and current metrics.

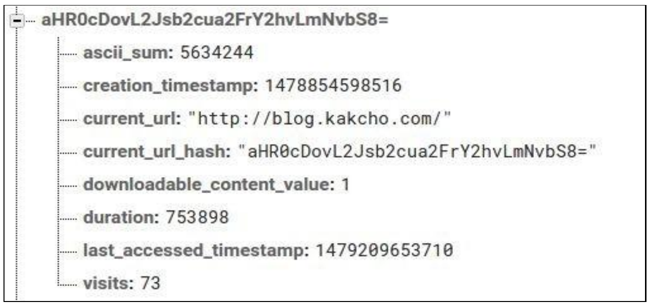
BM25 score values - first ordered list: BM25 score values for context matching along with the query set are shown in the figure 4 below:



```
QID    DID    Doc Len         Score
0      0      205             6.85895429668    NH-BM25
0      1      199             6.85895429668    NH-BM25
0      2      58              6.19349427941    NH-BM25
0      3      59              4.1188300278     NH-BM25
0      4      141             4.1188300278     NH-BM25
0      5      218             3.84424656401    NH-BM25
0      6      52              2.81306566725    NH-BM25
0      7      34              2.81306566725    NH-BM25
0      8      117             2.81306566725    NH-BM25
0      9      16              2.81306566725    NH-BM25
```

Figure 4: First order list created after BM25 scoring

Dataset for clickstream analysis- The database for clickstream analysis consists of a collection of hashed URLs in JSON format. Each hashed URL contains the statistics related to that URL like visits, duration, ASCII sum, etc. as shown in the Figure 5 below:



```
⊟ aHR0cDovL2Jsb2cua2FrY2hvLmNvbS8=
    ascii_sum: 5634244
    creation_timestamp: 1478854598516
    current_url: "http://blog.kakcho.com/"
    current_url_hash: "aHR0cDovL2Jsb2cua2FrY2hvLmNvbS8="
    downloadable_content_value: 1
    duration: 753898
    last_accessed_timestamp: 1479209653710
    visits: 73
```

Figure 5: A database record of clickstream engine

Final Ordered List - The final ordered list obtained in the order or normalized scores is as follows:

S. No.	URL	Importance Value
1.	http://blog.kakcho.com/	13.83
2.	http://blog.kakcho.com/fashion-in-himachal-pra...	10.03
3.	http://blog.kakcho.com/fashion-in-haryana/	9.58
4.	http://blog.kakcho.com/fashion-in-jammu-and-k...	5.91
5.	http://blog.kakcho.com/fashion-in-maharashtra/	5.51
6.	http://blog.kakcho.com/about-us/	5.02
7.	http://blog.kakcho.com/subscribe/	4.83
8.	http://blog.kakcho.com/amity-fashion-college/	3.78
9.	http://blog.kakcho.com/wp-content/uploads/201...	3.74

Figure 6: Final ordered list ordered by their importance values

Order Graph-This graph plots the Importance Values (IVs) of the top 20 URLs ordered by their calculated score.

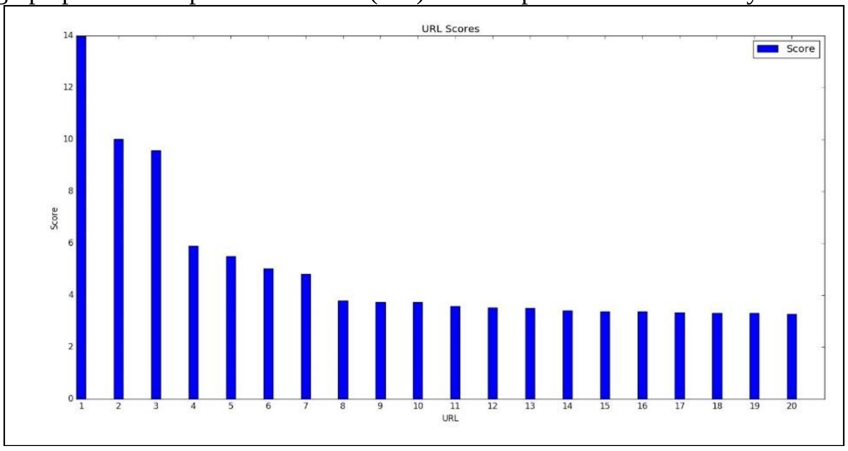


Figure 7: Graph with Web pages sorted according to IVs from most relevant to least relevant

Performance Metrics- The following metrics were obtained by running the architecture on the aforementioned hardware.

METRIC	VALUE
Crawler	
Crawled links	1054 links
Execution time	3 minutes 40 seconds.
Downloader	
Execution Time	579.57289052009583seconds.
corpus.txt size	3.9MB
BM25 Analysis	
corpus.txt	3.9 MB
queries.txt	62 bytes
Execution Time	0.417417049408 seconds
Importance Value Calculations	
Fetch time from firebase	1.5554604530334473 seconds
Time to calculate top 20 and make graph	1.4693260192871094

Figure 8: Performance Metrics of the Web crawler

7 Comparison with similar existing crawlers

- **Scalability:** The implemented crawler is much more scalable than the crawler proposed in [2]. It supports Load balancing, indexing and real time Clickstream collection that the previously proposed crawler does not address.
- **LessInvasiveness:** The previously proposed crawler uses Server Logs for clickstream data however, server logs contain user session information which should not be shared with anyone. OAuth access token and OAuth token are highly sensitive data which can be a serious threat to security. The implemented crawler uses JavaScript to obtain metrics and doesn’t take any sensitive information therefore making it less invasive and more secure.
- **Completeness of data:**One of the drawbacks of using Web server log files is that on a repeat visit to a site, the server won’t recognize the request as it may be processed through the browser’s cache. Thus, the user’s interaction with the site won’t be picked up rendering the data incomplete. However, by using a script written in JavaScript, no such issue arises as every visit to that Web page updates the analytics data.
- **Relevance of Data:** Web logs were built for primarily to collect server activity, not data for analytics. Over time they have been enhanced to collect more and more data and store it with some semblance of sanity to meet the needs to developers and decision makers whereas JavaScript tags were developed with the intent to collect clickstream data for analysis. In as much they are much more focused about what they do and only collect data that they need (though admittedly not all the JavaScript tags running around are smart and they do collect unnecessary data).
- **Easier access to data**: The data from Web server log files is usually difficult to read, complicated to transport to a central server, and impossible to digest in raw format. However, using JavaScript tags make it easier to access relevant data as per the needs of the developer without the hassles of managing big and bulky Web server log data.

8 Conclusion

The proposed architecture uses JavaScript based clickstream analysis for a less invasive and more secure clickstream analysis solution. The JavaScript and JSON have been minified to have very less effect on Webpage load times. The WebCrawler integrates Okapi BM25 Ranking function and Clickstream analysis to calculate ImportanceValue(IV) of each crawled Web page. The clickstream analysis calculates permanent importance of a Web page and is therefore an important metric. A real-time database ensures that all values are updated in real time without any delay for increased performance. The crawler internally uses Map-Reduce jobs for processing thus further improving the speed of the entire process. Thus, a secure, scalable, and polite Clickstream based parallel focusses Web crawler.

9 Future Scope

In this document, various issues in web data extraction have been discussed. Although the implemented crawler extracts the data from surface Web pages successfully, the clickstream engine relies on those websites that have agreed to embed our JavaScript in their Webpages. There is need of a more robust and generic platform for collecting clickstream usage metrics which are standardized by an established authority to make clickstream analysis, a mainstream factor in page ranking. Also, improvements can be made in the system, in the near futurefor crawling deep Web Webpages to find and index hidden Web pages residing in the dark net.

10 References

[1] S. Sharma and P. Gupta, "The anatomy of web crawlers," *International Conference on Computing, Communication & Automation*, Noida, 2015, pp. 849-853.doi: 10.1109/CCAA.2015.7148493

[2] I-Hsien Ting, C. Kimble and D. Kudenko, "UBB Mining: Finding Unexpected Browsing Behaviour in Clickstream Data to Improve a Web Site&#146;s Design", *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*.

[3] F. Abkenari and A. Selamat, "A Clickstream-based Focused Trend Parallel Web Crawler", *International Journal of Computer Applications*, vol. 9, no. 5, pp. 1-8, 2010.

[4] P. Giudici, Applied Data Mining, Web Clickstream Analysis, Wiley Press, 2003. Chapter 8, ISBN:0-470-84678-X.

[5] K. Sparck Jones, S. Walker and S. Robertson, "A probabilistic model of information retrieval: development and comparative experiments", *Information Processing & Management*, vol. 36, no. 6, pp. 779-808, 2000.

[6] H. Lu, D. Zhan, L. Zhou and D. He, "An Improved Focused Crawler: Using Web Page Classification and Link Priority Evaluation", Mathematical Problems in Engineering, vol. 2016, pp. 1-10, 2016.

[7] A. Seyfi and A. Patel, "A focused crawler combinatory link and content model based on T-Graph principles", Computer Standards & Interfaces, vol. 43, pp. 1-11, 2016. doi:10.1016/j.csi.2015.07.001

[8] F. Ahmadi-Abkenari and A. Selamat, "An architecture for a focused trend parallel Web crawler with the application of clickstream analysis", *Information Sciences*, vol. 184, no. 1, pp. 266-281, 2012. doi: 10.1016/j.ins.2011.08.022

[9] Tomala, K., Plucar, J., Dubec, P., Rapant, L. and Voznak, M. (2013). The Data Extraction Using Distributed Crawler Inside Multi-Agent System. AEEE, 11(6). doi:10.15598/aeee.v11i6.867

[10]M. Nasri and M. Sharifi, "Load Balancing Using Consistent Hashing: A Real Challenge for Large Scale Distributed Web Crawlers," *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on*, Bradford, 2009, pp. 715-720. doi: 10.1109/WAINA.2009.96

[11]D. Karger, E. Lehman, T. Leighton, M. Levine, D.Lewin, R. Panigrahy, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web", In Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso Texas, 1997, pp. 654-663. doi: 10.1145/258533.258660

[12]D. L. Quoc, C. Fetzer, P. Felber, E. Riviere, V. Schiavoni, and P. Sutra, "UniCrawl: A practical geographically distributed web Crawler," 2015 IEEE 8th International Conference on Cloud Computing, Jun. 2015. DOI: 10.1109/CLOUD.2015.59

[13]X. Chen, W. Li, T. Zhao and X. Piao, "Design of the Distributed Web Crawler", Advanced Materials Research, vol. 204-210, pp. 1454-1458, 2011. DOI: 10.4028/www.scientific.net/AMR.204-210.1454

[14]M. Sunil Kumar and P. Neelima, ""Design and Implementation of Scalable, Fully Distributed Web Crawler for a Web Search Engine"", International Journal of Computer Applications, vol. 15, no. 7, pp. 8-13, 2011. DOI: 10.5120/1963-2629

[15]M. Bahrami, M. Singhal and Z. Zhuang, "A cloud-based Web crawler architecture," *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, Paris, 2015, pp. 216-223.
doi: 10.1109/ICIN.2015.7073834