sciendo

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **19**, No 2
Sofia • 2019 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2019-0021

Optimized Focused Web Crawler with Natural Language Processing Based Relevance Measure in Bioinformatics Web Sources

S. R. Mani Sekhar[1], *G. M. Siddesh*[2], *Sunilkumar S. Manvi*[1], *K. G. Srinivasa*[3]

[1]*School of C & IT, Reva University, Bangalore, India*
[2]*Deptarment of Information Science & Engineering, Ramaiah Institute of Technology, Bangalore, India*
[3]*National Institute of Technical Teachers Training and Research, Chandigarh, India*
E-mails: manisekharsr@gmail.com siddeshgm@gmail.com ssmanvi@reva.edu.in srinivasakg@gmail.com

***Abstract*:** *In the fast growing of digital technologies, crawlers and search engines face unpredictable challenges. Focused web-crawlers are essential for mining the boundless data available on the internet. Web-Crawlers face indeterminate latency problem due to differences in their response time. The proposed work attempts to optimize the designing and implementation of Focused Web-Crawlers using Master-Slave architecture for Bioinformatics web sources. Focused Crawlers ideally should crawl only relevant pages, but the relevance of the page can only be estimated after crawling the genomics pages. A solution for predicting the page relevance, which is based on Natural Language Processing, is proposed in the paper. The frequency of the keywords on the top ranked sentences of the page determines the relevance of the pages within genomics sources. The proposed solution uses a TextRank algorithm to rank the sentences, as well as ensuring the correct classification of Bioinformatics web page. Finally, the model is validated by being compared with a breadth first search web-crawler. The comparison shows significant reduction in run time for the same harvest rate.*

***Keywords*:** *Focused crawler, Data extraction, Natural Llanguage Processing, Topical Crawler, TextRank, Distributed Crawler, Master-Slave architecture, Bioinformatics.*

1. Introduction

The Internet has revolutionized the world by empowering people with data accessibility. The Internet help and share information among people. It has emerged uncontrollably to become a huge collection of information. The collective information present on the internet can reveal facts that are unfathomable to an individual. Information present on the internet is an excellent source because of the

hierarchical structuring. The hyperlink network information simplifies the task of finding related information. It is necessary to tape and extract the mass potential information available on the internet. This function will be performed by the Crawler. The word Crawler was invented by C h a k r a b a r t i, D o m and B e r g [8].

The concept of extracting information from the internet is not novel. Search engines, Data Miners, and surveyors have used automated retrieval information since beginning of twentieth century. The automated software browses the web and extracts data from the internet. Such software is referred to as web crawler or web spider. The web crawler [10] here-in-after simply referred as crawler. The crawler is central concept behind all search engines. Search engines use crawlers to index mass data but they can also be used as a tool for data extraction.

There is always a specific purpose for data mining. The analysis of data is performed so as to prove or to discover facts that were earlier unknown. When the analysis is specific, the information extracted must also be relevant to the context. Extracting relevant data from the internet is a task performed by special crawlers referred as Focused Web Crawler. The Focused Web Crawler browses the internet accumulating information relevant to this concept.

The Focused Web Crawler starts extraction at a seed page and traverses all of its hyperlinks. On crawling child pages the crawler discovers related information. On an average, a page has hundreds of hyperlinks. Each of the hyperlinks is a probable source of relevant information. It is evident that the number of pages the crawler can crawl increases exponentially. The efficiency of data extraction can be improved by running them into multiple instances. When a cluster of crawlers run on a single topic, the coordination is a matter of concern. This paper proposes a master-slave architecture where slaves are the crawlers and master coordinates. Master collects and extracts information. While abstracting the coordinating, it makes crawlers into discrete units that can be added or removed from a system on demand. The dispensable nature of the slaves along with the central coordination from the master enables the system to absorb the varying difference in response time. Even crawling a single page is a time-consuming task, it is necessary to select the most relevant pages among the lot. The Focused Web Crawler predicts the probable relevance of the page from its parent. An irrelevant page would not have links to a page of relevance. The more relevant the page is – the higher the chance it's child pages will also be. The measure of relevance of a page is also an area of concern. The relevance of a page is measured with respect to a set of keywords. The keywords define the context of the search. One of the simplest methods of measuring relevance is the frequency of the keywords in the page.

[15] developed machine ontologies specific domain information extraction from web sources which contain huge domain groups and their interfaces of specific domain. [16] Developed a computational model for wrapped data by incorporating SQL and relational methodology and also allow the users to perform query and to optimize information subsequently. However, this method lacks the scalability and capability. Whereas the author [17] presented a novel method for extraction of complex structured web sources by integrating SQL and relational methodology, but the system works on manual coding of source.

BLAST [11] application provides a web based interface, which helps in identifying DNA and protein libraries in large dataset. It also helps the biologist in finding the protein sequences, subsequently allowing researcher in identifying a duplicate record.

As datasets are increasing day by day, accessing to Bioinformatics web sources becomes a challenging task for genomics. The significance of retrieving data from these sources requires advance computation techniques. Identification of these dataset faces many challenges. The key challenges are:

- Finding and evaluating new Bioinformatics Web sources.
- To provide integrated access point for multi-model database.
- Creation of effective navigation path and pattern.
- Provide a single access point for genomic data source.

The proposed design and implementation in this paper adopt Natural Language Processing to rank the importance of a sentence in the page. This paper uses frequency of the keyword in the top sentences as a measure of relevance.

1.1. Focused Web Crawler

Focused crawler is an automated mechanism to efficiently find web pages relevant to a topic on the web. Focused crawlers are built on crawl specific portion of the web starting from a set of initial pages referred to as the seed set. The applications of such crawlers are diverse. Focused crawlers help update the index of a search engine, data extraction and distributed processing of the complete web. The crawlers are used for personalized or community search engines, web databases, and business intelligence. A Focused Web Crawler has components that download pages given by the URL (Universal Resource Locator) in a priority queue, processing of the pages and handling to download contents. There are multiple methods to implement focused web crawlers. These methods depend upon their strategies as well as to measure their relevance. A Focused Crawler uses a set of keywords as reference to determine the relevance.

The measure of performance of a Focused Web Crawler is defined by its harvest ratio. The harvest ratio is calculated by its precision and recall. In pattern of recognition or information of retrieval precision is defined as the fraction of retrieved instances that are relevant. The recall is the fraction of relevant instances that are retrieved. Positive predictive value and sensitivity are synonyms of precision and recall respectively. These two factors help to determine the relevance to set of keywords in a page. The next equation shows harvest ratio in simple non-statistical terms can be denoted as the number of relevant pages crawled to the number of crawled pages:

$$\text{harvest ratio} = \frac{\text{number of relevant pages}}{\text{number of crawled pages}}. \quad (1)$$

1.2. TextRank

On the surface, Graph based ranking algorithms score a particular vertex V that represents an entity such as a webpage by finding the number of incoming edges to *V* which represent the hyperlinks leading to the webpage. To calculate the score of a

particular vertex V_i in a graph depends on the number of vertices j with links to V_i and the scores of each incoming vertex V_j. This procedure is repeated recursively over the whole graph. The score of vertex V_i is

$$s(Vi) = (1 - d) + d * \sum_{j \epsilon \mathrm{In}(Vi)} \frac{1}{|\mathrm{Out}(Vj)| S(Vj)}. \quad (2)$$

Mihalcea and Tarau [1] have proposed the use of graph ranking algorithms on the text. The method involves four major steps. The entire passage is divided into text units such as words, sentences, etc. The type of text unit depends on the application. The text units represent the vertices of the graph. Next, the relations between the text units are identified. These represent the edges of the graph. The graph ranking algorithm is used over the whole graph and the scores of the vertices are calculated. Finally, the vertices are sorted according to their score values. The higher the score of a text unit the more relevant it is within the context of the given passage. The solution proposed in this paper uses this technique in order to find the relevance of sentences within web pages.

Rest of the paper is divided into five sections, section second illustrates the related work, subsequently section third and fourth describe the proposed model and algorithm in detail. Next, section fifth discusses the result achieved by incorporating proposed model. Finally, sixth section concludes the presented work.

2. Related work

The Service Class Description system (SCD) [12] preforms a classification of web sources by focusing only on capability and conversational of semantic, which could lead to latency problem. Furthermore, the amount of bioinformatics sources and their rate of change faces many difficulties in prediction of relevant pages. Our present work automatically solves the above stated issues by making slave parallel.

In [13] a metadata classes are used in identifying bioinformatics web sources, subsequently they are also used for analytic solution that helps in mapping bioinformatics source back to that description. The key features of whole section are organized in the framework of BLAST web source. Finally, this report could be helpful in determining the random source. The efficiency of the system can be further increased by introducing master slave architecture in parallel with Natural Language Processing (NLP).

ExALG [14] proposed the solution to infer the pattern for information withdrawal from a collected HTML pages, the system does not provide any description of the website class. A model has been defined which encode the input values by template functions and subsequently excerpts the output sequences which are encoded in the pages. In the proposed work, the above stated problem is solved by introducing content and keyword search mechanism.

Fuzzy and weight concepts are used in [3] to determine the importance of web pages with respect to the topics. Here the focused web crawler computes the significance between the given topic and web pages using appropriate relevant measures. The information is retrieved using ontology graph and fuzzy inference system for the specific domain. The efficiency can be further increased by incorporating master slave architecture.

To improve the effectiveness of prediction in Focused Web Crawlers a learning model is proposed in [4]. They used Naïve Bayesian method for prediction, here the authors have used only four attributes such as parent page, URL keywords, anchor words, and adjacent text for computation.

[5] Proposed a methodology that helps in extracting significant and hidden pages by integrating information related to rank and semantics. The information contained within the thresholds is considered for computation. The prediction can be further by incorporating master slave architecture in parallel.

3. Proposed distributed Web Crawler

A distributes crawler is a parallel computing architecture which can download the source of the websites from different parts of the world. The servers and the crawlers interact by running different protocols, operating different servers, different allocated bandwidth, and handling different user traffic. Multiple factors play in the latency of server response time. A parallel system of coordinating crawlers is necessary to improve the efficiency of crawlers. If the servers are slow, it blocks the requesting crawler. The peer crawlers absorb the load while the crawler waits for the server to respond. The work distribution is not the complete solution as a focused crawling system requires coordination among crawlers. The potential problems of a distributed crawler are:

i. Duplicate-crawls;
ii. Prioritizing crawl frontier;
iii. Duplicate data;
iv. Scalability.

Master-Slave model [2]. Dividing the task into two modules make crawlers into independent units that can be added on or removed from the system. A Master-Slave model proposes a centralized coordinator and a set slaves under the coordinator. The design assigns the task of crawl frontier prioritization, data accumulation and slave management to the master. A slave system is a unit with dedicated internet connection which on demand from the master downloads a page and runs data extraction, sends it to the master and scores the page on its relevance. This architecture allows the number of slaves to be flexible. Fig. 1 depicts proposed Master-Slave architecture for web crawler.

Master maintains the crawl frontier and is responsible for avoiding duplicate crawls. Prioritization of the crawl frontier is a task that is essential to the efficiency of the crawler. The hyperlinks are prioritized based on the parent's rank. The assumption here is that there is a higher probability that a page is more relevant if its parent is relevant. The hierarchical structure of the internet ensures pages to have information related to its neighbours. The proposed system assumes a priority queue based on selection of the crawl frontier. The theoretical proof as to why this priority queue is that the most efficient can be derived logically. The relevance score can only be calculated after downloading a page. The previous assumption assumes that the child node stores information related to its parent. The related information stored in hierarchical structure implies that the relevance of a page is relative to its parent.

The seed page is considered to be of high relevance to the topic being crawled. The assumption further implies that we crawl from the seed page, the more partial the relevance is. The partiality of relevance is the case where the page may be relevant to a subset of the keywords. A crawl with the keywords (money, business, country, president) may be referring to a topic on the crash of the market after the presidential election. A partial relevance is that a page is relevant to money and country only. The page might be the comparison of the economy of different countries. The page is relevant in some sense but this content is not what the crawler should prioritize. Such a condition, hereafter referred to as partial relevance grows with the depth of the search. These two facts, the relevance score is relative to its parent and the fact that the more partial the relevance grows with depth, implies that a greedy approach is the best suited. A pure greedy approach like priority queue is the best approach. If the depth were limited, first search would be inefficient as the depth increases the more partial its relevance is.

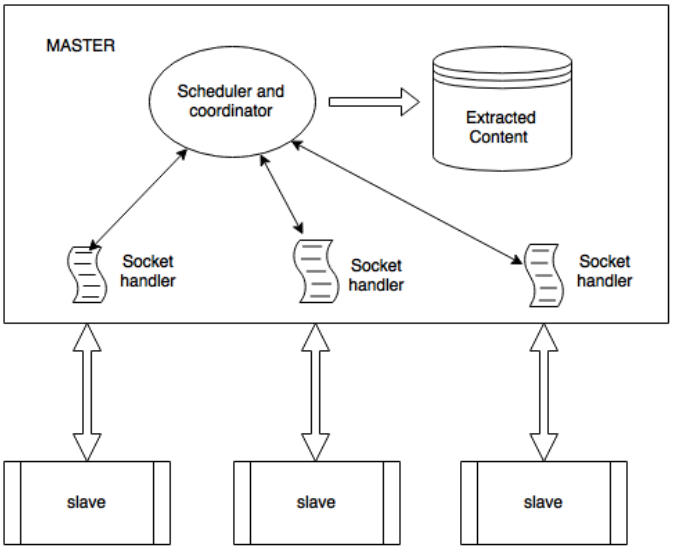

Fig. 1. Proposed Master-Slave architecture for Bioinformatics web sources

The architecture of multiple expendable slaves to a single master illustrated in Fig. 1 is latency and fault tolerant. Slaves while downloading pages may enter busy waiting while the rest of the crawl frontier is crawled by its peer systems. Fig. 1 illustrates the components of the master communicating with the slave machines. The master draws up multiple kernel threads each running a socket handler. The slave communicates over a persistent TCP connection. The initiated thread lives until the slave replies or until the event of a timeout. The master thread waits a until receipt of the reply, while the peer threads process the growing crawl frontier.

The scheduler [2] is a simple priority queue based on ordering of the crawl frontier. The crawl frontier is constantly updated by the slaves. The crawl front being a resource accessed by multiple threads is given mutual exclusion using semaphores. A simple First Come First Serve (FCFS) scheduling is used to give access to the scheduler. The priority in the priority queue is the score of the hyperlink being pushed

into the queue. The coordinator is the process which maintains and controls the slaves. The coordinator is responsible for configuring the master machine with the available slaves. The coordinator maintains a slave availability system. Being once initialized all slaves' sets are to be available. The coordinator maintains mutual exclusion by monitoring the allotment of the slaves. Once the coordinator is requested for a free slave, it responds by giving the requested thread the slave or a busy signal indicating all the slaves are occupied. As the threads complete the processing of a hyperlink using a slave, the thread calls the coordinator to release, which in turn would put the slave back into the available pool. Then slaves are allotted in an FCFS manner, as there is no priority among the several threads that are being processed.

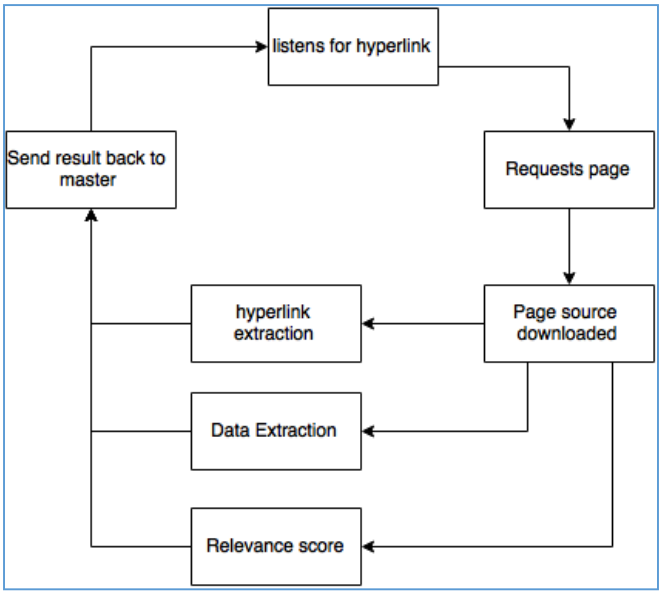

Fig. 2. State diagram of slave system of proposed solution

Slave listens scrape requests that are received from the master on a pre-decided port. The master needs to be configured with the set of slaves it works with. On initialization the master reads the configuration file and adds into a queue of available slaves. The slave performs three specific tasks that have varying completion time. The threading and parallelization of these processes over multiple slave systems absorb the variable latency. The tasks of a slave system are to download the source of the requested hyperlink, to extract the content and hyperlinks and to calculate the relevance score of the page. The relevance score, set of hyperlinks and the content are sent back to the master. Fig. 2 illustrates the flow of slave states in the proposed solution.

3.1. Relavance score

In order to find relevance two factors are used namely, the "keyword exists" factor and the TextRank frequency factor. For each given keyword, it is found out whether the "keyword exists" within the document. This occurrence value is divided by the

total number of input keywords to give the “keyword exist” factor. The extracted content is then tokenized into sentences. Each sentence is converted into a “bag of words” which is a collection of words and their respective frequencies in each sentence. The bag of words is represented as a sparse matrix wherein each column is a word and each row is a particular sentence in the web page. Each element of this matrix is the frequency of a particular word in the column within a particular sentence given in the row. The process of creation of the sparse matrix is called Vectorizing. To remove the effect of common and non-essential words in each sentence, the matrix is normalized using TF-IDF. This is done because words such as “the”, “a”, “and”, etc., do not represent the content central to a particular piece of writing. In order to calculate the similarity score of each word the normalized matrix is multiplied with its transpose. In the similarity matrix each row as well as column represents a sentence and each element has a value between 0 and 1. A value of one represents sentences that are exactly the same and 0 represents sentences that have absolutely nothing in common.

A semantic graph is created using the similarity matrix and TextRank is run on this graph. Once the relevance score of each sentence is calculated using the TextRank algorithm, the top most 40% relevant sentences are chosen. The number of occurrences of each input keyword within the top most 40% is calculated in relevant sentences. The resultant frequency factor is called the TextRank frequency factor. The product of the “keyword exist” factor and the TextRank frequency factor gives the overall relevance score of a particular web page. In this work Slave operating algorithm, Crawl operating algorithm, scoring algorithm and Master operating algorithm have been used for identification of Bioinformatics web sources.

4. Proposed work

In the proposed solution Natural Language Processing is adopted to rank the importance of a sentence in the page. Subsequently the model use frequency of the keyword in the top sentences as a measure of relevance of genomic web source.

4.1. Slave operating algorithm

Algorithm 4.1 summarize the Slave operating algorithm. The given input is splited into small chunks using split function, subsequently it extract the required information from the machine using crawl scrape. Finally, the computed result sended to the Master module.

Algorithm 4.1. Slave operating algorithm
Input: Request
Output: Link and Keyword
Step 1. def handler (request):
Step 2. data = request.receive()
Step 3. data = data.split(“||”)
Step 4. hyperlink = data[0]
Step 5. keywords = data[1]
Step 6. result = crawl.scrape (hyperlink, keywords)

Step 7. master.send(result)
Step 8. def slave(port, host):
Step 9. server = SocketServer.TCPserver((host, port), handler)
Step 10. server.server_forever()

4.2. Crawl operating algorithm

Algorithm 4.2 discusses the Crawl operating algorithm. The link and keyword is supplied as an input to the scrape function, these values are used to extract the source hyperlink. Subsequently the source hyperlink is extracted. The extracted hyperlink is used in the extraction of contents. Later the score is computed based on the keyword and content.

Algorithm 4.2. Crawl operating algorithm

Input: Link & keyword
Output: Content, score and link
Step 1. **def** scrape(hyperlink, keyword)
Step 2. source = fetch_source (hyperlink)
Step 3. **if**(source)
Step 4. parse_obj = parse(source)
Step 5. hyperlinks = extract_hyperlink(parse_obj)
Step 6. content = extract_content(parse_obj)
Step 7. score = score(content, keywords)
Step 8. return hyperlinks + content + score
Step 9. else
Step 10. **return** error

4.3. Scoring algorithm

Algorithm 4.3 illustrates the Scoring algorithm. After classifying the prescribed word for the given sentence and keyword the word count function automatically increments the counter, this process is repeated continuously for the given sentence. Subsequently the score is computed using text rank methodology by performing Tokenizing, Vectorising and Normalization

Algorithm 4.3. Scoring algorithm

Input: Sentence and Keyword
Output: Score
Step 1. def wordCount(sentence, keywords):
Step 2. **for** each word “w” in keywords:
Step 3. **for** each word “k” in sentence:
Step 4. **if** w == k
Step 5. count[w]++
Step 6. **for** each word in keywords:
Step 7. **if** count[word] >= 1
Step 8. result++
Step 9. return result
Step 10. def score(content, keywords)
Step 11. keyword_present = wordCount(document, keywords)

Step 12. keyword_exist_factor = keyword_present / number_of_keywords
Step 13. sentences = tokenize(content)
Step 14. sparse_mat = vectorize(sentences)
Step 15. normalised_mat = normalize_TFIDF(sparse_mat)
Step 16. similarity_mat = normalised_mat Xtranspose(normalised_mat)
Step 17. graph = semantic_graph(similarity_mat)
Step 18. score = textRank(graph)
Step 19. relevant_sentences = number of sentences * 0.40
Step 20. text_rank_frequency= wordCount(relevant_sentences, keywords)
Step 21. final_score = text_rank_frequency * keyword_exist_factor
Step 22. **return** final_score

4.4. Master operating algorithm

Algorithm 4.4 illustrates the master operating algorithm. The priority queue system is the scheduler; scheduler works closely with the coordinator or the slave system. The schedulers can only crawl on the availability of a slave. The scheduler prioritizes the crawl frontier using a simple priority queue. The greedy approach to the prioritization is logically proved to be the best approach because of the phenomena "partiality of relevance".

Algorithm 4.4. Master operating algorithm
Input: Seed, Keyword
Output: Score
Step 1. def master():
Step 2. Seed = getSeed()
Step 3. Keywords = getKeywords()
Step 4. initialize_slaveSystem()
Step 5. pQueue(seed, keywords)
Step 6. def pQueue(seed, keywords):
Step 7. priorityQueue.push(seed)
Step 8. **if** crawl_depth<depth_limit:
Step 9. break
Step 10. **else** pQueue.empty():
Step 11. slave = getFreeSlave():
Step 12. assign_slave: Hyperlink = priorityQueue.get()
Step 13. crawled_list.append(hyperlink)
Step 14. T = createThread(slave_call)
Step 15. T.daemon = true
Step 16. T.start()
Step 17. def slave_call():
Step 18. create_socket(slave_port)
Step 19. send_slave_hyperlink()
Step 20. Data = receive_response()
Step 21. Data = data.split("||")
Step 22. Content = Data.content
Step 23. Hyperlinks = Data.hyperlinks

Step 24. Score = Data.score
Step 25. For_Hyperlink: priorityQueue.push(link, score)
Step 26. Database.write(Content)

4.5. Slave System

Slave system is a coordinator system that maintains the availability of the slaves. The slaves are blocked on request and allotted to the requested thread. The thread calls the getFreeSlave function to request a free slave. On completion of the task, the thread unblocks the slave by calling the unblock function. Algorithm for slave system is described in Algorithm 4.5.

Algorithm 4.5. Slave System algorithm
Input: File
Output: Slave
Step 1. def initialize_slaveSystem():
Step 2. Read configuration file
Step 3. Append slaves to slave list
Step 4. def block_slave(index):
Step 5. Slave_list(index).blocked = true
Step 6. def unblock_slave(index):
Step 7. slave_list(index).blocked = false
Step 8. def getFreeSlave():
Step 9. check slave in slave_list:
Step 10. **if**(slave.block==false): block_slave(index(slave))
Step 11. return slave

5. Results and discussion

Fig. 3 enumerates the efficiency of the proposed solution by measuring operating time in comparison with single and multi-slaves. The optimized operating time absorbs the varying latency in downloading the page. The multiple slaves are expendable. As a slave enters busy waiting for downloading the page source, the rest of the slaves can tackle the crawl frontier. As shown in Fig. 3, when the depth is 25 operating time of multi slave and single slave is 8 & 140, whereas when the depth is 55 operating time of multi slave and single slave is 40 & 250. Meanwhile when the depth is 80 operating time of multi slave and single slave is 60 & 360. Finally, when the depth is 97 operating time of multi slave and single slave is 80 & 425. Table 1 shows successful and un-successful crawled data using master slave architecture. It consists of four columns, column one represent the serial number. Second column show the keyword used for web crawling bioinformatics data source. Column third indicate the total hit for the specific keyword, subsequently last column show the status whether the stated keyword is successfully crawled or not, here positive value represent successfully crawled data and negative value represent unsuccessfully crawled data.

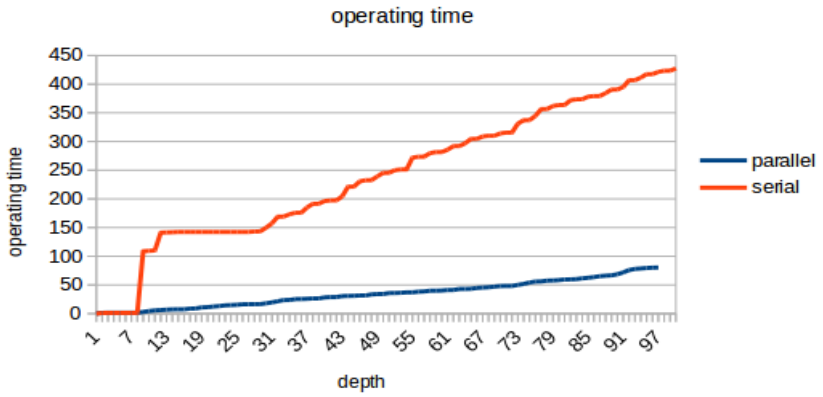

Fig. 3. Operating time comparison for single and multi-slaves

Table 1. Web sites classified using Master Slave architecture

Site No	Keyword	Total hits	Status
1	Protein	110	Positive
2	HIV	80	Positive
3	Twit	0	Negative

The architecture has proved itself simple to install and to debug. The variable latency is also solved by parallelized slaves. The architecture is very scalable as the crawling operating time decreases with the increase in number of slaves. The extraction of content from the source is done by filtering the visible content, as this process is also time consuming. The above problem is optimized by tokenizing the xml. The scoring of the pages is also optimized without leveraging the accuracy of the score. The initial screening test eliminates the pages that are not worth scoring. These pages are ignored. Thus, as discussed above multi-slave architecture outperform when compared with single slave architecture.

6. Conclusion

It is being proved that web is a necessary tool for researchers, developers and scientists. The proposed solution highlights the need of web sources in Bioinformatics domain; subsequently which illustrates the process of overcoming the data management issues that ascends during identifying the genomic web sources. The proposed solution uses Master Slave architecture with NLP concept for effective extraction of bioinformatics web sources. In addition, it provides a solution for latency problem by parallelizing slaves. Meanwhile the scalable nature of the architecture is perfect for large scale crawling for integrated genomic sources. As scoring the relevance of the page is an important aspect of the crawler, the scoring has to be optimized by either leveraging the accuracy or the operating time. The scoring is optimized by improving the accuracy. Finally, ranking the sentences helps the scoring system to work accurately. Ideally in the future tokenizing the sentences can be optimized to suit the focused crawler. The result shows that optimization of

the focused crawler is achieved better when the computation is performed in parallel when compared to serial computation.

References

1. M i h a l c e a, R., P. T a r a u. TextRank: Bringing Order into Texts. University of North Texas, UNT Digital Library, 2004.
2. W a n, Y., H. T o n g. URL Assignment Algorithm of Crawler in Distributed System Based on Hash. – IEEE International Conference on Networking, Sensing and Control (ICNSC'2008), 2008, pp. 1632-1635.
3. J a l i l i a n, O., H. K h o t a n l o u. A New Fuzzy-Based Method to Weigh the Related Concepts in Semantic Focused Web Crawlers. – IEEE, 2011, pp. 23-27.
4. M e j d l, S., A. A l t h a g a f i., D u n r e n. Improving Relevance Prediction for Focused Web Crawlers. – In: 11th International Conference on Computer and Information Science, IEEE/ACIS,2012, pp.161-166.
5. P a v a n i, K., G. P. S a j e e v. A Novel Web Crawling Method for Vertical Search Engines. IEEE, 2017, pp. 1488-1493.
6. L o k h a n d e, K. P., S. S. H o n a l e, H. N. G a n g a v a n e. Web-Crawler Using Priority Queue. – International Journal of Research in Advent Technology, Vol. **2**, 2014, No 2.
7. D i x i t, A., et. al. URL Ordering Policies for Distributed Crawlers: A Review. 2015, arXiv preprint arXiv:1611.01228.
8. C h a k r a b a r t i, S., B. D o m, M. V a n den B e r g. Focused Crawling – A New Approach to Topic Specific Web Resource Discovery. – Elsevier Science B.V. Computer Networks, Vol. **31**, 1999, No 11, pp. 1623-1640.
9. C a s t i l l o, C. Effective Web Crawling. Ph. D. Thesis, University of Chile. Retrieved 2010-08-03, 2004.
10. D u b e y, J., D. S i n g h. A Survey on Web Crawler. – International Journal of Electrical, Electronic and Computer System, Vol. **1**, 2013, Issue 1. ISSN: 2347-2820.
11. A l t s c h u l, S. F., W. G i s h, W. M i l l e r, E. W. M e y e r s, D. J. L i p m a n. Basic Local Alignment Search Tool. – Journal of Molecular Biology, Vol. **215**, 1990, No 3, pp. 403-410.
12. A n n e, H., H. N g u, D. R o c c o. Terence Critchlow David Buttler, Automatic Discovery and Inferencing of Complex Bioinformatics Web Interfaces. – Journal of World Wide Web, Vol. **8**, 2005.
13. R o c c o, D., T. C r i t c h l o w. Automatic Discovery and Classification of Bioinformatics Web Sources. – Journal of Bioinformatics, Vol. **19**, 2003.
14. A r a s u, A., H. G a r c i a-M o l i n a. Extracting Structured Data from Web Pages. – In: Proc. of ACM/SIGMOD Annual Conference on Management of Data, 2003, pp. 337-348.
15. M o d i c a, G., A. G a l, H. M. J a m i l. The Use of Machine-Generated Ontologies in Dynamic Information Seeking. – In: 9th International Conference on Cooperative Information Systems, CoopIS2001, 2001, pp. 433-448.
16. H a a s, L., P. S c h w a r z, P. K o d a l i, E. K o t l a r, J. R i c e, W. S w o p e. Discoverylink: A System for Integrating Life Sciences Data. – IBM Systems Journal, Vol. **40**, 2001, No 2.
17. D a v i d s o n, S. B., G. C. O v e r t o n, V. T a n n e n, L. W o n g. BioKleisli: A Digital Library for Biomedical Researchers. – International Journal on Digital Libraries, Vol. **l**, 1997, No l, pp. 36-53.

Received: 22.02.2018; Second Version: 22.11.2018; Third Version: 28.12.2018; Accepted: 14.02.2019