Distributed Web Crawlers using Hadoop

Pratiba D
Assistant Professor, Department of Computer Science and Engineering,
R V College of Engineering, R V Vidyanikethan Post, Mysuru Road
Bengaluru, Karnataka, India.
Orcid Id: 0000-0001-9123-8687

Shobha G
Professor, Department of Computer Science and Engineering,
R V College of Engineering, R V Vidyanikethan Post, Mysuru Road
Bengaluru, Karnataka, India.

LalithKumar H
Student, Department of Computer Science and Engineering,
R V College of Engineering, R V Vidyanikethan Post, Mysuru Road
Bengaluru, Karnataka, India.

Samrudh J
Student, Department of Information Science and Engineering,
R V College of Engineering, R V Vidyanikethan Post, Mysuru Road
Bengaluru, Karnataka, India.

Abstract
Web Crawler is a software, which crawls through the WWW to build database for a search engine. In recent years, web crawling has started facing many challenges. Firstly, the web pages are highly unstructured which makes it difficult to maintain a generic schema for storage. Secondly, the WWW is too huge and it is impossible to index it as it is. Finally, the most difficult challenge is to crawl the deep web. Here we are proposing a novel web crawler, which uses Neo4J, HBase as data stores. It also applies Natural Language Processing (NLP) and machine learning techniques to resolve the above-mentioned problems.

Keywords: NoSQL, Neo4J, Hbase, Natural Language Processing, Reinforcement learning, Deep web

INTRODUCTION
Web Crawler is an application that feeds a search engine with web content. A search engine searches over the crawler's indexing of the WWW. The data store used for storage, query and retrieval of the indexed content plays an important role in determining the crawler's performance. There is a need for a schema-less storage and easier query techniques with the ever increasing structural complexity of web pages. The database needed here must be schema-less, horizontally scalable and should provide higher performance for queries dealing with large amount of data. It should also be able to represent the relationship between web pages for faster computation of most appropriate results. So, by using HBase as the data store to store the indexed pages and using a graph database like Neo4J for storing the page, file relationships and hierarchy, the data store needs can be sufficed.

With the increasing size of web content, it is very difficult to index the entire WWW. So, efforts must be put to limit the amount of data indexed and at the same time maximize the coverage. To minimize the amount of data indexed, Natural Language Processing techniques need to be applied to filter and summarize the web page and store only the essential details. By doing so, a search can yield the most accurate and appropriate results based on the prominence of words instead of their mere existence. Deep web refers to the sections of the WWW, which are difficult to index. Dynamic web pages which can be generated only through a form and pages tailored based on the access context and navigation sequence are just the main problems encountered while crawling deep web. To maximize the coverage of WWW, a web crawler can never neglect the deep web. By applying Natural Language Processing and reinforcement learning to gain domain specific knowledge of a website and by gaining insights about the domain, the web crawler can build intelligent queries against forms to navigate through the website. In this paper, we will deliberate upon three methods to improve the efficiency of a web crawler.

Web Crawler is an integral part of any search engine. As the number of pages in increasing massively, there must be a sophisticated search engine to track and traverse this huge data and provide user with a better results for their queries. Hence, a web crawler keeps track of all the web pages inside the search engine by traversing through the internet and downloads as many web pages as possible and indexes the relevant web pages. These indexes are maintained up to date along with the path to the relevant web pages.

A web crawler is an automated script, which is used to extract
data from the WWW in a periodical way. Web crawler is an application that feeds a search engine with processed web content which makes the process of ranking easier. A search engine searches over the crawler's indexing of the WWW. Search engine use web crawling application to periodically appraise their indexed web content or indexes of other sites. Crawlers consume resources on the servers they crawl and often visit without site’s approval. It is important to note that a crawler needs to take into consideration issues of scheduling, load, and politeness delay when large collections of pages are indexed. Public sites have mechanisms to inform the crawlers that the content is not allowed to be crawled.

Web Crawler has a distinctive life cycle which starts with set of the seeds (URLs) which is originally extracted from the user or from a web page. It then crawls deeper (Download) into the sub URLs created in the main page. The procedure begins with downloading the corresponding web pages and then extracting the entire sub URLs found in the page, followed by population of the frontier which is a queue data structure. The downloading process is abled by using Hyper Text Transfer Protocol (HTTP). The downloaded pages are indexed. WebCrawler stops at the point where no more seeds exist in the frontier.

Types of Web Crawlers

Web Crawlers are classified [1] based on the methodology used to crawl the web pages. Some of the types are listed below:

Incremental Web Crawlers: The traditional crawlers are used to replenish the old documents with new ones to augment its collection. On the other hand, an incremental crawler updates an existing set of indexed pages instead of re-indexing the pages on subsequent crawls that are invoked in a fixed time. An incremental crawler enhances the present set of pages by visiting them frequently, based on the frequency of how often page change. This technique exchange less important pages with more important ones. The advantage of incremental crawler is that only appreciated data is provided to the user, thus saving the network bandwidth and achieves data enhancement.

Hidden Web Crawlers: Crawlers, which are used to crawl the deep web, are called Hidden Web Crawlers. Hidden web refers to the abundant data inside the databases. These data are hidden behind the query interface and are not directly accessible to the search engine.

Focused Crawlers: Focused Crawler is also called as Topic Crawler. This web crawler minimizes the downloads by downloading only the appropriate and significant pages that are related to each other. It avoids crawling unimportant paths by limiting the area of interest of a search to a particular topic. This deals with crawling the deep web to an extent. This type of crawl is economically feasible with respect to the hardware and network resource which reduces the network traffic and downloads.

Parallel Crawlers: Web pages are crawled in parallel with the help of multiple threads in order to fasten the speed of indexing and to counter the problem of indefinite wait on a page. This crawler consists of several crawling processes known as C-procs which runs on network terminals. This type of crawling system plays a vital role in downloading documents in a sensible amount of time.

Distributed Web Crawlers: Sometimes even a process with multiple threads is not enough to index large amount of web pages. In such a situation, a cluster of many workstations are used to crawl the web. A dominant server maintains the interaction and management of the nodes as it is geologically dispersed. This type of crawling is called Distributed web crawler [2]. It mainly routines page rank algorithm for the better efficacy and quality search. The advantage of this crawler is that it protects against system crashes.

Breadth First Crawlers: This method crawls in breadth first fashion [3]. All the web pages of lower depth are crawled before a web page of higher depth is crawled.

Traditional Web Crawlers

In the year 1993, four web crawlers were introduced. They were like Wide Web Wanderer, Jump Station, World Wide Web Worm, and RBSE spider. They collect data, information and statistics about the web using a set of seed URLs. These crawlers iteratively download URLs and update their repository of URLs through the downloaded web pages.

In the year 1994, two new web crawlers, WebCrawler and MOMspider crawler [4] were introduced. These web crawlers started collecting data and statistics of the web along with the concept of politeness and black-list to traditional web crawlers. Simultaneously 15 links can be downloaded and hence web crawlers are considered to be the first parallel web crawler. The number of indexed pages were increased from 110,000 to 2 million from WWW to Web Worm. These crawlers arises scalability issues.

Brin et al., [5] addressed the issue of scalability by introducing a large scale web crawler called Google. Google was the large web crawler that addressed the problem of scalability in several ways. Firstly, it is optimized to reduce disk access time through techniques like compression and indexing. Secondly, probability of a user visiting a page was calculated using a Page Rank algorithm. Page Rank algorithm computes the probability of a user visiting a page by considering the number of links that point to the page along with the style of those links. This approach optimize the resource available to the web crawler by reducing the rate of the web crawler visiting unattractive pages. Using this technique, Google achieved high freshness. The framework of Google used master-slave architecture with the master being
the URK server that dispatches URLs to a set of slave nodes. The slave nodes retrieve the assigned pages by downloading them from the web. The implementation of Google supported 100 downloads per second.

The issue of scalability was further addressed by [6] using a tool called Mercator. Mercator addressed two problems that is extensibility and scalability of web crawlers. To address extensibility, it took advantage of a modular Java-based framework. By doing this, it allowed third-party components to be integrated into Mercator. To address the problem of scalability, Mercator tried to solve the problem of URL-Seen. The URL-Seen means that the URL is seen or not. This trivial problem consumes a lot of time as the URL list grows. Mercator increased the scalability of URL-Seen by batch disk checks. By doing this, hashes of discovered URLs got stored in RAM. Once the hashes grows beyond a certain limit, the lists was equated with the URL’s stored on the disk. The list of URL’s on the disk was updated. Using this technique, the second version of Mercator crawled 891 million pages. Mercator got integrated into AltaVista in 2001.

IBM introduced WebFountain [7]. It was a fully functional web crawler which was not only used to index the web but also used to create a local copy of it. This local copy got updated regularly as often as WebFountain visited the page and a copy of it was kept indefinitely in the local space. With WebFountain, the major components like scheduler was distributed and the crawling process continued which resulted in increased local copies of web. With all these features and the most efficient technology such as Message Passing Interface (MPI), made WebFountain a scalable web crawler with high acceptance rate. As the web grew, WebFountain simulated 500 million pages and scaled twice its size every 400 days.

In 2002, Polybot addressed the problem of URL-Seen scalability by enhancing the batch disk check technique. The Red-Black tree technique was used to keep the URL’s, once it is scaled to a certain limit, it was merged with a sorted list in main memory. With this technique, Polybot managed to scan 120 million pages [8].

UbiCrawler [9] used a peer-to-peer (P2P) approach to deal with URL- seen problem which was very consistent in hashing the URL and distribution among the web crawler nodes. With this model, there was no centralized units calculating URL, seen or not, but when a URL was discovered it is passed to the node responsible to answer the test. The nodes doing this calculation, maps it to the nodes in the list by taking hashes of the URL. UbiCrawler could download at a rate of 10 million pages per day with just five 1GHz PCS and 50 threads.

In 2002, pSearch crawler used two algorithms called P2P Vector Space Model (pVSM) and P2P Latent Semantic Indexing (pLSI) to crawl the web on a P2P network. It also addressed the scalability issue by using Distributed Hash Tables (DHT) routing algorithms. To calculate the relation between queries and the documents, VSM and LSI used vector representation. In 2003, these techniques were used to scale up certain tasks such as clustering of contents and bloom filters. Loo, Cooper and Sailesh Krishnamurthy [10] addressed the issue of scalability of web crawlers and used the technique to partition URLs among the crawlers. With this technique, a requests of 8 lakh pages from 70,000 web crawlers were done in 15 minutes. This indicates that this work was done with high speed communication medium.

Exposto, Macedo, Pina, Alves and Rufino [11] augmented partitioning of URLs among a set of crawling nodes in a P2P architecture by taking into account servers geographical information. Such an augmentation reduced the overall time of the crawl by allocating target servers to a node physically closest to them. In 2008, an extremely scalable web crawler called IRLbot used a quad-CPU AMD Opteron 2.6 GHz server and it crawled over 6.38 billion web pages in 41.27 days. This web crawler addressed the issue of URL-Seen by breaking it down into three sub-problems: CHECK, UPDATE and CHECK+UPDATE. It used a robust framework called Disk Repository with Update Management (DRUM). DRUM segmented the disk into several buckets to optimize the disk access. URLs are mapped to a disk bucket and the DRUM allocates corresponding buckets on the RAM. The buckets on the RAM are accessed in a batch mode, as soon as the buckets on the RAM is filled. DRUM stored a large number of URLs on the disk by doing batch mode access, as well as the two-stage bucketing system so that its performance is not comprised and would not degrade as the number of URLs increases.

Applications of Web Crawlers

Some of the applications of web crawlers [1] are discussed below:

- Web Crawlers are used by the search engines to gather up-to-date information.
- A crawler can be implemented over a relational database to provide text based search of the columns
- Web Crawlers are used for automating Web site maintenances of web pages, validation of HTML syntax and also verifying the validity of the links.
- Crawlers can be used to efficiently mine required information from many websites Ex. Harvesting email addresses for advertisements.
- Linguists use web crawlers to perform textual analysis. They address the internet to determine what words are commonly used.
- Web crawlers are used in biomedical applications such as finding relevant literature on a gene.
- Researchers in the market field use web crawlers to
determine and analyze trends in given market. The results of this help to make recommendation system.

**Challenges of Web Crawling**

There are many challenges [12] faced by the web crawlers based on the crawling technique. Some of them are listed below:

- **Missing Relevant Pages:** Most of the times, the relevant pages are missed by crawling only the expected pages that gives immediate benefit.

- **Maintaining Freshness of Database:** Many HTML pages consist of information that gets updated on regular basis. The crawlers have to download such pages and update them in the database up-to-date which makes the crawler slows and impacts on the internet traffic. The freshness of the collection of pages vary depending on the strategies used by the crawler to crawl the web. But keeping the collections fresh is a big challenge.

- **Network Bandwidth and Impact on Web Servers:** Some of the crawling techniques downloads irrelevant web pages that leads to consumption of network bandwidth. The bandwidth is also consumed more for the maintenance of multiple crawlers used to maintain the database. If the servers are visited by the crawlers every time for the information, then there is an impact on the performance of the web servers. The crawlers sometimes trigger abuse alarms in web servers when the user tries to access a web page repeatedly.

- **Absence of particular context:** While crawling, if a particular topic is not available, then it may download large number of irrelevant pages.

- **Overlap:** Sometimes, the same pages will be downloaded multiple times and increases the redundancy.

- **Communication Overhead:** In few crawling techniques, the contributing crawling nodes must exchange URLs to coordinate the overall crawling job. The communication overhead is defined in terms of the exchanged URL’s per downloaded page. This overhead must be minimized.

- **Improve quality of the local collection:** The crawlers must replace less important pages with more important ones as some new pages may be more important than the existing pages in the collection.

- **Dangling up of web crawlers:** Sometimes the web crawlers are hung up in case of late or no response from web servers.

- **Resolving of DNS:** The DNS servers crash due to substantial loads.

**APPLICATIONS OF NOSQL DATABASES IN WEB CRAWLERS**

Web crawling collects information and resources from internet and stores it in the database and constructs a search engine for the users. It requires an abundant storage space. The relational database stores the data in terms of rows and columns which results in poor horizontal scalability and high hardware cost when the data storage increases [13]. With the increase in the resource information, the traditional relational database can no longer suffice the requirement to query information.

The traditional databases are widely used in all the types of database applications. However, the relational databases are not able to cope up with the continuous development of web applications. This gave birth to a new generation of database which is a NoSQL Database.

NoSQL as Not Only SQL, is the definition of non-relational data storage. From 2007 to the present, a dozen of the popular products have emerged, such as Google’s BigTable, Apache’s HBase, FaceBook’ Cassandra, 10gen’ MongDB, Yahoo!’ PNUTS etc.

**Shortcomings of Relational Databases**

The shortcomings are explained below:

- **Data Storage:** As web crawlers collects large amount of data among the huge web pages, storing such data in tabular format is not feasible.

- **Machine Performance:** A web crawler needs a large number of tables that are related. The relational databases lack in machine performance for queries involving many tables.

- **Physical Storage Consumption:** A web crawler in particular involves processing and querying of large amount of data. In a RDBMS, for example an operation like join would consume a lot of physical storage.

- **Slow extraction of information from data:** There is a lack of flexibility in the way the data can be stored and organized. The stored data cannot be interpreted in the case of RDBMS.

- **Maintenance cost:** Since huge information is extracted from the web crawlers, storing these data for the relational database requires additional servers with more storage space that leads to high maintenance cost.

- **Scalability:** Since the data collected from the web
servers are large, maintaining such data in distributed data management system in the relational database is not supportive and hence results in poor horizontal scalability.

Advantages of NoSQL Databases

NoSQL databases have a wide range of advantages. NoSQL databases were built to eliminate the limitations of traditional relational databases. They provide superior performance and are easily scalable. The data model in NoSQL databases counter the disadvantages of the relational model.

The advantages of NoSQL include being able to provide:

- Flexibility in the data model used in the crawler.
- Easier and faster query methods compared to the RDBMS.
- High availability and partition tolerance for the indexed data to be accessed by the web crawler that meets the requirements of many practical internet applications with huge data storage.
- Ease of understanding of the indexed data.
- Web crawlers provide the user with the most relevant pages while searching.
- Reduces data consistency and integrity constraints.
- Good horizontal scalability with low cost as the databases have the high performance of read and write operations.

HBase as a Data Store for indexed documents

HBase is an Apache Hadoop based open source project [14]. HBase is a distributed column based multidimensional map structure on top of the Hadoop file system. Its data model is similar to Google’s BigTable that provides fast random access to huge amount of structured data. HBase is used in companies like Facebook, Twitter, Yahoo and Adobe.

HBase for indexed documents supports schema-free design and hence designing the structure beforehand is not required as it can be modified at run time. It supports variable schema that is, any number of columns can be added and removed dynamically.

HBase supports blob storage of indexed data. The web repository stores and manages a large collection of web pages. These web pages are compressed and are assigned with the Docid, len and URL. The document index keeps track of each document ordered by Docid. The information stored in the entry are current document status, pointer to the repository, checksum and other various statistics. If a document has been crawled, then it contains a pointer to the variable width file called doc info, which contains its URL and title. Once the index is completed, they are stored in the memory and updated repeatedly as new content is crawled. The indexed words being the key and the web page details as the columns. The query is easier as it is a key value data store and provides fast and efficient results when compared to many other NoSQL databases. Hence HBase can be used as a document store for dumping of indexed data.

HBase provides auto failover, reliability, flexibility in the data model for the web crawler, high scalability with low cost of hardware and replicate set of data [15]. And also it makes the query fast through order-preserving sharding which simply means that tables are dynamically distributed by the system when they become too large. The configurations for auto sharding can be easily set.

Neo4J as a Data Store for Page-File Relationships

As Hbase is not a great choice when it boils down to representing relationships between the web pages and the hierarchy, Neo4J [16] is used for data representation. Neo4j is an open source NoSQL graph database. It is a completely transactional database (ACID) which stores structured data as graphs that consist of nodes which in turn are connected by relationships. It is durable and persistent. It provides concurrency control, transaction recovery and high availability [17]. It is provided with a disk-based, native storage manager optimized for storing graph structures and it results in extreme performance and scalability. Neo4j can handle billions of nodes and relationship on a single machine and also can be scaled out across multiple machines for high availability. It consist of a powerful graph traversal framework and query language called Cypher that allows user to efficiently access data by expressing graph patterns.

- Relationship store format for the nodes

Since node keeps only a reference to the single relationship, relationships are organized in doubly-linked lists, which makes all relationships of some node traversable from this node. Each relationship is a part of two linked lists: a list of relationships of the first node, from which this relationship starts, and a lists of relationships of the second node, at which this relationships ends.

- Indexes

Neo4j supports indexing of nodes and relationships by labels and property values, by doing this it allows retrieving all nodes with given label and/or property value faster than via full scan of all nodes in the database. Apparently production implementation of indexes is fully
delegated to Lucene engine. Entity identifiers are stored in Lucene documents with fields, corresponding to the indexed labels and property values. Lucene is able to search documents by individual fields and combinations of fields, empowering complex queries on Neo4j level.

Neo4J on the other hand provides the following advantages.

- The connected nodes can be represented using Neo4J
- More connected data can be easily and quickly retrieved/traversed/navigated.
- It is easy to represent semi-structured data.
- Cypher Query Language (CQL) is the query language command of Neo4j which is in human readable format and very easy to learn.
- It delivers simple and powerful data model.
- As it is very easy to retrieve its adjacent nodes or relationship, it does not require complex joins to retrieve connected/related data

APPLICATIONS OF NLP IN WEB CRAWLERS

Webpage Summarization

The increasing amount of data on the WWW has made it impossible for the data to be indexed as it is. Therefore efforts need to be put in order to minimize the amount of data to be indexed. Text summarization [18] is one such method which can be employed in order to index only the essential features of a web page. Summarization can be done by rewriting the original text or by mining key sentences from the text. Our preferred method is to do mining of key sentences which works better in most of the cases.

The script needs to identify web URLs to extract full content. The main HTML document is mined after excluding headers, footers, advertisements and side bars by using a template removal process. Then DOM parser is used to extract article data which includes meta description, title, actual content, top image and top video. The final step of summarization is done by the following criteria:

- Presence of the title words
- Sentence length
- Position of the sentence
- Presence of keywords and their intersection
- Frequency of occurrence

Deep web crawling using NLP and reinforcement learning

As server-side programming and scripting languages, such as PHP and ASP, got momentum, more and more databases became accessible online through interacting with a web application. The applications often delegated creation and generation of contents to the executable files using Common Gateway Interface (CGI). In this model, programmers often hosted their data on databases and used HTML forms to query them. Thus a web crawler cannot access all of the contents of a web application merely by following hyperlinks and downloading their corresponding web page. These contents are hidden from the web crawler point of view and thus are referred to as deep web.

Lawrence and Giles [19] estimated that 80 percent of web contents were hidden. Later, Bergman and M.K [20] recommended that the deep web content is 500 times larger than what outsiders through hyperlinks. As more companies are moving their data to databases, the size of the deep web is growing rapidly.

Deep Web is indexed by search engines only in a small fraction. He, Patel, Zhang and Chang [21] tested one million IPs and crawled these IPs looking for deep webs through HTML form elements. The study also defined a depth factor from the original seed IP address and constrained itself to depth of three. Out of the tested IPs, 126 deep web sites were found. These websites had 406 query gateways to 190 databases. Based on these results with 99 percent confidence interval, the study estimates that at the time of that writing, there existed 1,097,000 to 1,419,000 database query gateways on the web. Further study projected that Google and Yahoo search engines visited only 32 percent of the deep web. The study also estimated that 84 percent of visited object overlap between two search engines and hence combining the two search engines does not improve the percentage of the visited deep web.

Information retrieval from the deep web in the second generation meant interacting with HTML forms. To retrieve the hidden information in the deep web, the web crawler would submit the HTML form many times and each time it is filled with different dataset. Thus the problem of deep web crawling was reduced to the problem of assigning proper values to the HTML form fields.

In designing a web crawler, an open and difficult question to answer is how to meaningfully assign values to the fields in a query form. As Barbosa and Freire [22] explained that it is easy to assign values to certain fields such as radio button, but the difficulty arises with the text box inputs. Many different proposals tried to answer this question as follows.

Raghavan and Garcia-Molina [23] suggested to fill up the text box inputs that are mostly dependent on human output.

Liddle, Embey, Scott and Yau [24] defined a method for detecting form element and formulate a HTTP GET and POST request using default values specified for each field. The proposed algorithm is not fully robotic as it asks the users input when required.
Barbosa et al. [22] proposed two phase algorithm to generate textual queries. The first stage collected a set of data from websites and they were used to assign weights to keywords. In the second stage, greedy algorithm was used to retrieve more contents with less number of queries.

Ntoulas, Zerfos and Cho [25] further defined three policies for sending queries to the interface: a random policy, a policy based on the frequency of keywords in a reference document, and an adaptive policy that learns from the downloaded pages. Given four entry points, this study retrieved 90 percent of the deep web with only 100 requests.

Lu, Wang, Liang and Chen [26] mapped the problem of maximizing the coverage per number of requests to the problem of set-covering and uses a classical approach to solve this problem.

In the Deep web crawling framework based on reinforcement learning [27], crawler is regarded as an agent and deep web database as the environment. The framework enables crawlers to learn a promising crawling strategy from its own experience and also allows for utilizing diverse features of query keywords.

At any stage the crawler perceives its state and selects an action. The environment responds by giving the crawler some response and changes its state to next level. Each query is encoded as a tuple using its linguistic, statistic and HTML features. The rewards of unexpected actions are evaluated by their executed neighbours. Because of the learning policy, a crawler can avoid using unpromising queries as long as some of them have been issued.

EXISTING METHODS

Traditional web crawlers crawl the web and generally store it in a RDBMS. These databases store data in the format of tables with each record being a tuple row. RDBMS are concerned about the consistency and integrity, thus has low efficiency. It does not support scalability and the infrastructure is also too expensive. Querying and maintaining the database is difficult. Web crawlers just filter the web pages for noise and index the rest of the contents. It is impossible to index all of the web. The deep web crawling is still blurry. There is not a fully efficient crawler.

Web Crawlers are important in search engines to give efficient results and to index the pages. The existing crawling is supported with parallel crawlers that uses threads and thread synchronization is needed which is an overhead. Using parallel crawlers, the efficiency was increased but it was not effective.

The distributed web crawlers overcame the drawbacks of threads i.e. parallel crawlers and increase the efficiency of web crawlers to a larger extent. By using Distributed crawlers, one can be able to achieve scale-in and scale-out easily along with Load balancing.

Most of the existing crawlers have scalability issues and do not provide efficient search results as it is not fast. As the amount of data increases in web pages, collecting and storing such large amount of data is not supported in the existing methods and querying from such storage results gives redundant results. And also, it consumes more time.

PROPOSED METHODS

Three methods are proposed to improve web crawling. Firstly, usage of NoSQL databases as data stores. HBase to be used for storing the indexed web data. Neo4J to be used as data store for metadata and representation of page-file relationships. Secondly, usage of NLP for summarizing text before indexing. Finally, usage of reinforcement learning to crawl deep web. The amount of data on WWW is drastically increasing. The size of memory needed to store the data will be constantly increasing and hence the cost involved. It is impossible for the data to be indexed as it is. Hence the data needs to be summarized using extraction/abstraction techniques before indexing it into the database.

Extraction techniques copy any important information to the summary such as key clauses, sentences or paragraphs, whereas abstraction involves paraphrasing sections of the source document. Abstraction can condense a text more strongly than extraction. Since they require use of natural language generation technology, which itself is a growing field, the programs to develop abstraction/extraction are harder to develop.

The drawbacks of using RDBMS in web crawling applications has been indicated in the previous sections. The main problem was about RDBMS not meeting the performance expectations of the web crawlers. The methodology proposed here advocates the use of NoSQL databases to suffice the performance expectations of web crawlers. It proposes a method of using 2 NoSQL databases to serve a search engine. Generally the problem that a search engine faces is the time taken by the database to respond with result to a search query. The time taken increases drastically with increase in the complexity of the query. The only way to reduce the amount of time invested on querying is by using a solution which stores in an easily query able form.

The search engines queries the indexed data twice firstly to search for the pages matching the search criteria and then a query to get the data like page file relationships and hierarchy in order to rank the pages in descending order of relevance. For the first kind of query that is, query on document index, it is best if the database is a key value store. Where in, a word will be a key and a number of columns which represent the existence of a word in a web page. Each of the columns contain details regarding the URL of the page, significance of the word and position of the word in the text. For the second
kind of query, the data is best served if it’s fed as a graph based datastore, so that it can be easily understood and can be easily traversed. Easy traversal of data in particular comes in handy while ranking pages in accordance to the rank of the links it contains. HBase database with its column key-value based data storage model is best suited in the case of document indexing. Along with the features of NoSQL databases, HBase gets with the properties and advantages that Hadoop environment offers. The database best suited to store file page relationships and hierarchy is a graph database which provides sophisticated query methods so that it gets easy for the search engine to run queries regarding to the relationship to other pages.

Data is stored in Neo4J in terms of the nodes and the relationship. They can store billions of nodes and relationships. These nodes and relationships are indexed by the labels and/or property values. Implementation of indexes is proxies to search engines where entity identifiers are stored in records with fields corresponding to the indexed labels and property values. Complex queries can also be resolved by traversing the nodes and provides the user with efficient results.

RESEARCH DIRECTIONS

Web crawling is an extremely vital application of the Internet which has an effect on the performance of the system based on the Database Selection. A Web Crawler is fault tolerant and scalable. One of the major research area, the study of which is required is about scalability of the system and the behavior of its components. A better hashing algorithm will also improve the efficiency and performance of the Web Crawler.

Further work could be done on creating a software which will bring together many crawling policies and keywords which can be utilized as a platform for researchers. Being able to perform reasonably good sentence detection as a part of an NLP approach along with mining of unstructured data can enable some pretty powerful text-mining capabilities, such as crude but very reasonable attempts at document summarization. There can be a lot of developments done in this area.

The research directions include indexing of files like images and documents such as pdf, word, ppt etc. AIM should be to make the crawler so powerful and intelligent that it should automatically generate articles which would be the summary of all the articles corresponding to that topic.

In the future, there is a need for studying ways to evaluate methodologies which will help to locate novel relevant web pages with similarity to the topic. One way would be to measure similarity between visited pages and an expanded query made of pages that are known to be related to the topic. The involvement of NoSQL databases would be a necessity in this case.

Another important research direction can be building more corpora [28] covering all major languages such as French. Gathering of tokens using the available online resources would be necessary for the same. Optimization of the program design which will ensure to get more data from all kinds of resources should be done.

There is also a need to interconnect the crawler with other linguistic and data processing tools to form a single system offering instant web corpora on demand. Other plans for the future can be analyzing the topics and genres of the downloaded texts and eventually balancing the downloaded content in this respect.

REFERENCES


[10] Loo, Cooper, Sailesh Krishnamurthy, (2004). Distributed Web Crawling over DHTs. ScholarlyCommons@Penn.


