*International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)*
*Volume 2, Issue 8, August 2015, PP 32-40*
*ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)*
*www.arcjournals.org*

# Natural Language Processing Technique for Information Extraction and Analysis

**T. Sri Sravya[1], T. Sudha[2], M. Soumya Harika[3]**

[1] M.Tech (C.S.E) Sri Padmavati Mahila Visvavidyalayam (Women's University), School of Engineering and Technology, Tirupati.

*srisravya1991@gmail.com*

[2] Head (I/C) of C.S.E & IT Sri Padmavati Mahila Visvavidyalayam (Women's University), School of Engineering and Technology, Tirupati.

*thatimakula_sudha@yahoo.com*

[3] M. Tech C.S.E, Assistant Professor, Sri Padmavati Mahila Visvavidyalayam (Women's University), School of Engineering and Technology, Tirupati.

*soumyamutyala@gmail.com*

**Abstract:** *In the current internet era, there are a large number of systems and sensors which generate data continuously and inform users about their status and the status of devices and surroundings they monitor. Examples include web cameras at traffic intersections, key government installations etc., seismic activity measurement sensors, tsunami early warning systems and many others. A Natural Language Processing based activity, the current project is aimed at extracting entities from data collected from various sources such as social media, internet news articles and other websites and integrating this data into contextual information, providing visualization of this data on a map and further performing co-reference analysis to establish linkage amongst the entities.*

**Keywords:** *Apache Nutch, Solr, crawling, indexing*

## 1. INTRODUCTION

In today's harsh global business arena, the pace of events has increased rapidly, with technological innovations occurring at ever-increasing speed and considerably shorter life cycles. This dissertation describes WebCrawler, the Web's first comprehensive full-text search engine. WebCrawler has played a fundamental role in making the Web easier to use for millions of people. Its invention and subsequent evolution, from 1994 to 1997, helped fuel the Web's growth by creating a new way of navigating hypertext: searching. Before search engines, a user who wished to locate information on the Web either had to know the precise address of the documents he sought or had to navigate patiently from link to link in hopes of finding his destination. As the Web grew to encompass millions of sites, with many different purposes, such navigation became impractical and arguably impossible. I built WebCrawler as a Web service to assist users in their Web navigation by automating the task of link traversal and creating a searchable index of the Web. Conceptually, WebCrawler is a node in the Web graph that contains links to many sites on the Web, shortening the path between searchers and their destinations. In addition to its impact on the Web, WebCrawler also made a number of contributions to computer science. Some of these contributions arose from new problems and solutions emanating from WebCrawler's novel mission, while others arose when theoretical research on hypertext, information retrieval, and distributed systems fell short in meeting the demands of the Web.

WebCrawler broke new ground as the first comprehensive full-text search system for the Web. So in this research we used two main parts i.e., Nutch and Solr for web crawling techniques. Apache Nutch is an open source Web crawler written in Java. By using it, we can find Web page hyperlinks in an automated manner, reduce lots of maintenance work, for example checking broken links, and create a copy of all the visited pages for searching over. That's where Apache Solr comes in. Solr is an open source full text search framework; with Solr we can search the visited pages from Nutch. Luckily, integration between Nutch and Solr is pretty straightforward. Apache Nutch supports Solr out-the-box, greatly simplifying Nutch-Solr integration. It also removes the legacy dependence upon both Apache Tomcat for running the old Nutch Web Application and upon Apache Lucene for indexing.

## 2. RELATED WORK

Over the years, Lucene and Solr established themselves as rock-solid technologies (Lucene as a foundation for Java™ APIs, and Solr as a search service). For instance, they power search-based applications for Apple iTunes, Netflix, Wikipedia, and a host of others, and they help to enable the IBM Watson question-answering system.

Over the years, most people's use of Lucene and Solr focused primarily on text-based search. Meanwhile, the new and interesting trend of big data emerged along with a (re)new(ed) focus on distributed computation and large-scale analytics. Big data often also demands real-time, large-scale information access. In light of this shift, the Lucene and Solr communities found themselves at a crossroads: the core underpinnings of Lucene began to show their age under the stressors of big data applications such as indexing the entire Twitter sphere. Our focus shifted to enabling easy scalability, near-real-time indexing and search, and many NoSQL features — all while leveraging the core engine capabilities.

Search engines are only for searching text, right? Wrong! At their heart, search engines are all about quickly and efficiently filtering and then ranking data according to some notion of similarity. Search engines also deal effectively with both sparse data and ambiguous data, which are hallmarks of modern data applications. Lucene and Solr is capable of crunching numbers, answering complex geospatial questions To demonstrate how a search engine can go beyond search, the rest of this section shows you an application that ingests aviation-related data into Solr.

### A. Exploring The Data:

With the Solr application up and running, you can look through the data and the UI to get a sense of the kinds of questions you can ask. In the browser, you should see two main interface points: the map and the search box. The Solr request that underlies near queries takes advantage of Solr's new spatial functionality, For instance, the randomized test framework (which is available as a packaged artifact for anyone to use) makes it easy for the project to test the interactions of variables such as JVMs, locales, input content and queries, storage formats, scoring formulas, and many more.

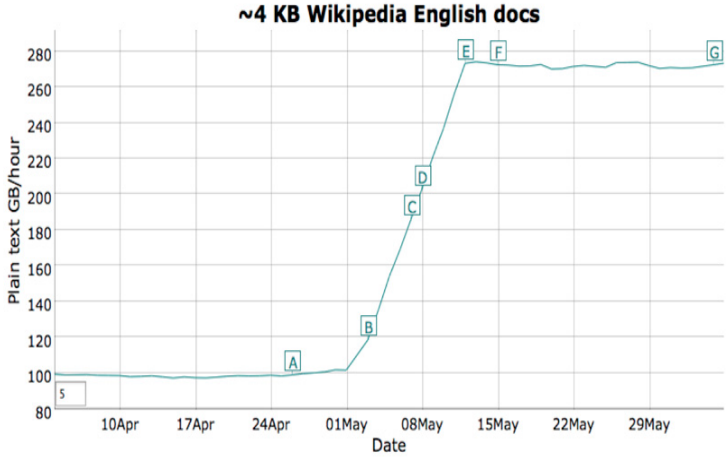


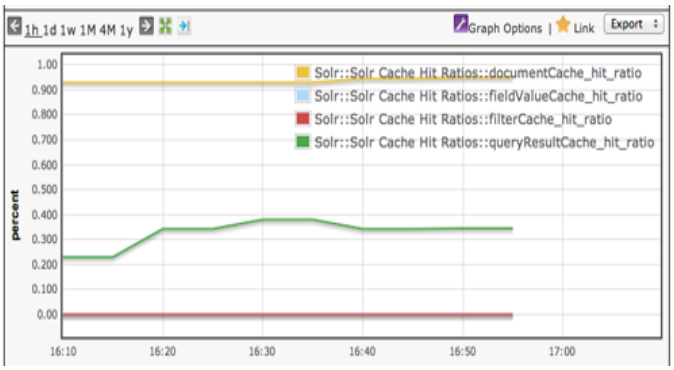

**Fig1.***Solr Indexing Performance*

Solr includes significant API changes and enhancements that are for the good of the engine — and that ultimately will enable you to do many new and interesting things. But upgrading from a previous version of Lucene might require significant effort, especially if you use any of the lower-level or "expert" APIs. (Classes such as Index Writer and Index Reader are still broadly recognizable from prior versions, but the way you access term vectors, for example, has changed significantly.)

### B. Flexibility

A Lucene segment is a subset of the overall index. In many ways a segment is a self-contained mini-index. Lucene builds its index by using segments to balance the availability of the index for searching with the speed of writing. Segments are write-once files during indexing, and a new one is created every time you commit during writing. In the background, by default, Lucene periodically merges smaller segments into larger segments to improve read performance and reduce system overhead.

Solr users can set and change these capabilities by modifying simple configuration items. Next-generation search-engine technology gives users the power to decide what to do with their data.




Search strategies are continuously improving their techniques and approach to provide more relevant, accurate information for a given query. They are working on different segments of search engine to improve the search results. Enhancing quality by improving crawling: Early search engines held an index of a few hundred thousand pages and documents, and received maybe one or two thousand inquiries each day. Today, a top search engine will index hundreds of millions of pages, and respond to tens of millions of queries per day. There is a series of modification in crawling mechanism to improve the quality of web information extraction:

**Keywords oriented strategy:** It start crawling pages based on keywords and then proceed with static list of URLs of web pages that contain keywords of query. ***Crawling with candidate URL structure:*** It uses contents of web pages pointing to current page, candidate URL structure, and other behaviours of siblings web pages in order to estimate the probability that a candidate is beneficial for a given crawl [4].

**Distributed crawling with static assignment**: There is a set of fixed rules that defines how new URLs will be assigned to the crawlers. For static assignment, a hashing function can be used to transform URLs into a number that corresponds to the index of the corresponding crawling process. Distributed crawling with dynamic assignment: In this approach a server works as a scheduler that assign URLs to crawlers at run time. Improvement on indexing segment: Different search engine uses different indexing mechanism to provide more relevant information in less amount of time. Different index mechanism is evolved in order to support search engine architectures to provide more relevant information. Types of indices include:

***Suffix tree:*** It supports linear time lookup. It built by storing the suffixes of words. The suffix tree is a type of trie. Tries support extendable hashing, which is importantfor search engine indexing.

***Inverted index:*** It stores a list of occurrences of each atomic search criterion as hash table or binary tree.

***Citation index:*** It stores citations or hyperlinks between documents to support citation analysis.

***Ngram index:*** Stores sequences of length of data to support other types of retrieval or text mining.

***Document-term matrix:*** Used in latent semantic analysis, stores the occurrences of words in documents in a two dimensional sparse matrix.

**Using temporal dimension to improve the quality of search:** The Web is not a static environment. It changes constantly. Quality pages in the past may not be quality pages now or in the future. Bringing new and quality pages to users is important because most users want the latest information.[3].

## 3. PROBLEM IDENTIFICATION

Although search engines are continuously modifying and adding new techniques to improve the quality of search but still search results are suffering with some problem.

Immaterial Results: Some times when we search for particular topic but we get results which are completely different from searched query. This problem exists because most words in natural languages that is they have multiple possible meanings or senses.

Redundant Content: Another problem associated with searching web is that sometimes contains links to redundant web pages which provides duplicate content. The presence of near duplicate web pages plays an important role in this performance degradation while integrating data from diverse sources. Web mining faces huge problems due to the existence of such documents.

**Some of the approaches stated are:**

**A.** ***Natural language Processing***:

Natural language processing (NLP) is a field of computer science that handles the interactions between computers and human (natural) languages. It allows computers to derive meaning from human or natural language input.

***B. Text mining:***

Text mining is also called as text data mining which is the process of deriving high-quality information from text. It analyses the text at character level and use mathematical expression to find the similarity of two pages. Text mining is a combination of the process of structuring the input text (usually parsing, and addition of some derived linguistic features and the elimination of others, and succeeding insertion into a database), deriving patterns within the structured data, and finally assessment and understanding of the output[5]. Removal of duplicate pages from search result will enhance the performance of strategy.

***C. Word Sense Disambiguation:***

Word Sense Disambiguation (WSD) is a branch of NLP which is the task of selecting the most appropriate meaning for a word, based on the context in which it occurs. Word sense disambiguation (WSD) can be used to overcome the problem of immaterial results[2]. All natural languages contain words that can mean different things in different contexts. WSD is an interior process in the natural language processing (NLP) chain. It is used in many applications such as text similarity check machine translation and information retrieval [3]. An approach that makes use of word sense disambiguation can achieve more accuracy in retrieving information for a given query.

## 4. PROPOSED METHODOLOGY

The proposed work describes an approach for web information retrieval that gives high quality result in terms of relevancy. This approach integrates multiple mechanisms to improve the quality of search engine result pages. First, it resolve the ambiguity in given search query that may occur due to words .This mechanism restrict the unwanted web pages from listing in search engine result pages. Second, it removes pages that have redundant contents from search engine result pages. Third, after performing these two levels of optimization it applies ranking technique of web pages. By this approach of search we will get search results of higher quality in terms of relevancy.

**Steps To Be Followed To Achieve The Proposed Method For Part 1 I.E., Nutch Crawling:**

**1. Customizing Crawl Properties:**

Nutch requires two configuration changes before a website can be crawled:

- Customize the crawl properties, where at a minimum, provide a name for the crawler for external servers to recognize.
- Set a seed list of URLs to crawl.

**2. Create a URL seed list:** A URL seed list includes a list of websites, one-per-line, which nutch will look to crawl. It will provide Regular Expressions that allow nutch to filter and narrow the types of web resources to crawl and download.

**3. Using Individual Commands for Whole-Web Crawling:** Whole-Web crawling is designed to handle very large crawls which may take weeks to complete, running on multiple machines. This also permits more control over the crawl process, and incremental crawling. It is important to note that whole Web crawling does not necessarily mean crawling the entire World Wide Web. We can limit a whole Web crawl to just a list of the URLs we want to crawl. Nutch data is composed of:

- The crawl database, or crawldb. This contains information about every URL known to Nutch, including whether it was fetched, and, if so, when.
- The link database, or linkdb. This contains the list of known links to each URL, including both the source URL and anchor text of the link.
- A set of segments. Each segment is a set of URLs that are fetched as a unit. Segments are directories with the following subdirectories:

- a *crawl_generate* names a set of URLs to be fetched
- a *crawl_fetch* contains the status of fetching each URL
- a *content* contains the raw content retrieved from each URL
- a *parse_text* contains the parsed text of each URL
- a *parse_data* contains outlinks and metadata parsed from each URL
- a *crawl_parse* contains the outlink URLs, used to update the crawldb

**4. Fetching:** To fetch, we first generate a fetch list from the database. This generates a fetch list for all of the pages due to be fetched. The fetch list is placed in a newly created segment directory. The segment directory is named by the time it's created.

**5. Invertlinks**: Before indexing we first invert all of the links, so that we may index incoming anchor text with the pages.[6-9]

**Steps to be followed to achieve the proposed method for part 2 i.e., Solr indexing:**

To select documents for the index, WebCrawler chose according to a number of criteria, often manually selected. First, we wanted to maintain some degree of breadth, so we identified the shortest URLs from each server for inclusion on the theory that the URLs with the fewest path components were most likely to be the more general pages on the server. We also identified the more popular pages from the entire Web for inclusion, reasoning that more popular pages were likely to be of higher quality. To determine popularity, we used data from the link table.

**6. Integrate Solr with Nutch:** The crawled data which is created by the url seed from nutch is linked with Solr for searching the require data.

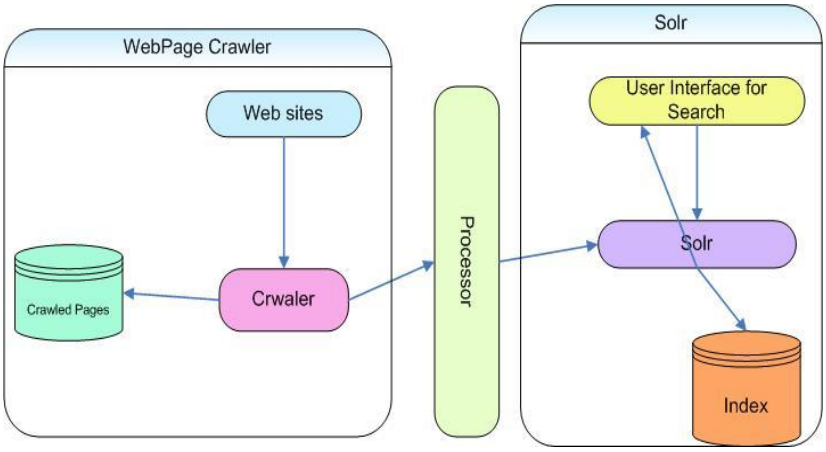



**Fig2.***Indexing Solr with Nutch*

**Proposed Method Architecture:**

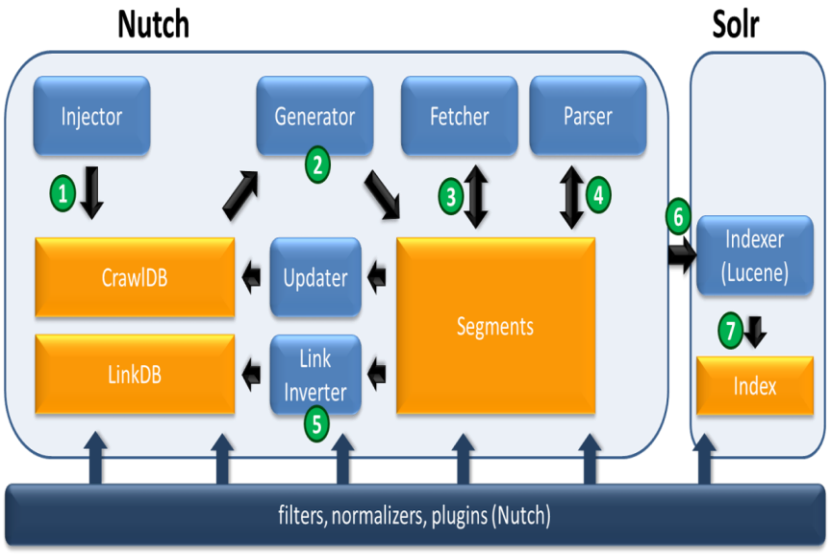



**Fig3.***Nutch-Solr Integration Architecture*

**Nutch** is an effort to build an open source web search engine based on Lucene and Java for the search and index component. Nutch is coded entirely in the Java programming language, but data is written in language-independent formats. It has a highly modular architecture, allowing developers to create plug-ins for media-type parsing, data retrieval, querying and clustering. The fetcher ("robot" or "web crawler") has been written from scratch specifically for the project.[6]

Advantages of Nutch over a simple fetcher are:

- highly scalable and relatively feature rich crawler
- features like politeness which obeys robots.txt rules
- robust and scalable - Nutch can run on a cluster of up to 100 machines
- quality - crawling can be biased to fetch "important" pages first

Many researchers have looked at web search technology over the last few years, including crawling strategies, storage, indexing, ranking techniques, and a significant amount of work on the structural analysis of the web and web graph. Highly efficient crawling systems are needed in order to download the hundreds of millions of web pages indexed by the major search engines.
In fact, search engines compete against each other primarily based on the size and currency of their underlying database, in addition to the quality and response time of their ranking function. A crawler for a large search engine has to address two issues. First, it has to have a good crawling strategy, i.e., a strategy for deciding which pages to download next. Second, it needs to have a highly optimized system architecture that can download a large number of pages per second while being robust against crashes, manageable, and considerate of resources and web servers.
Solr makes it easy for programmers to develop sophisticated, high-performance search applications with advanced features such as faceting (arranging search results in columns with numerical counts of key terms). Solr builds on another open source search technology: Lucene, a Java library that provides indexing and search technology, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities. Since Solr is a server, it is more common to run it in the background, especially on Unix/Linux.[7-10]

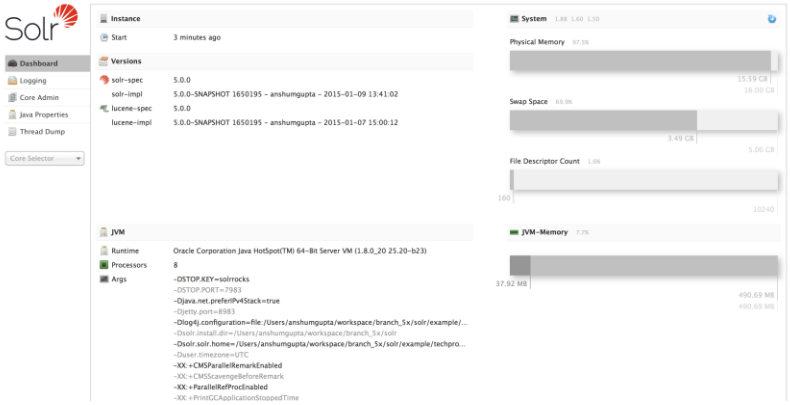
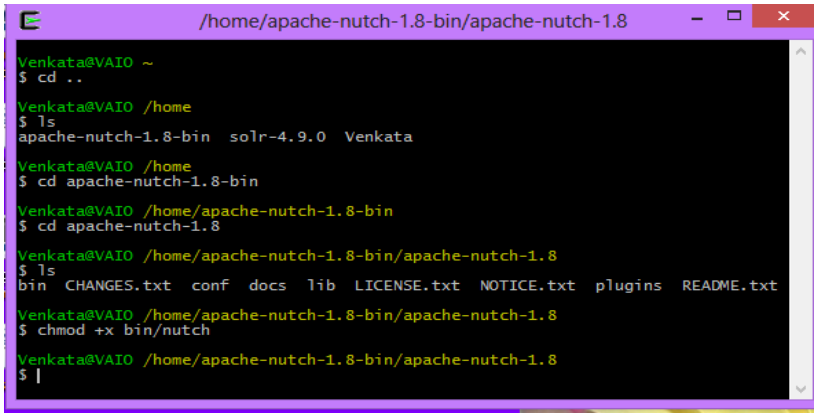
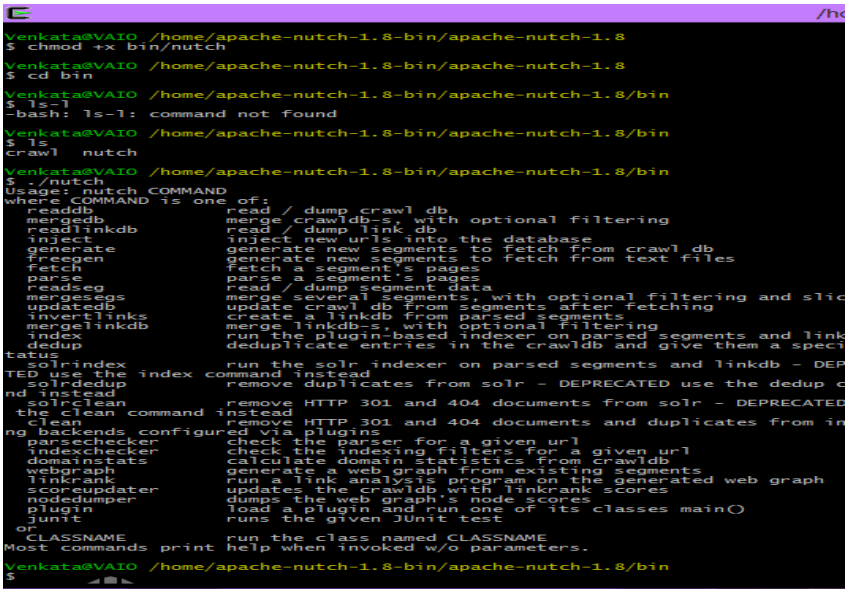



**Fig4.***Solr Interface*

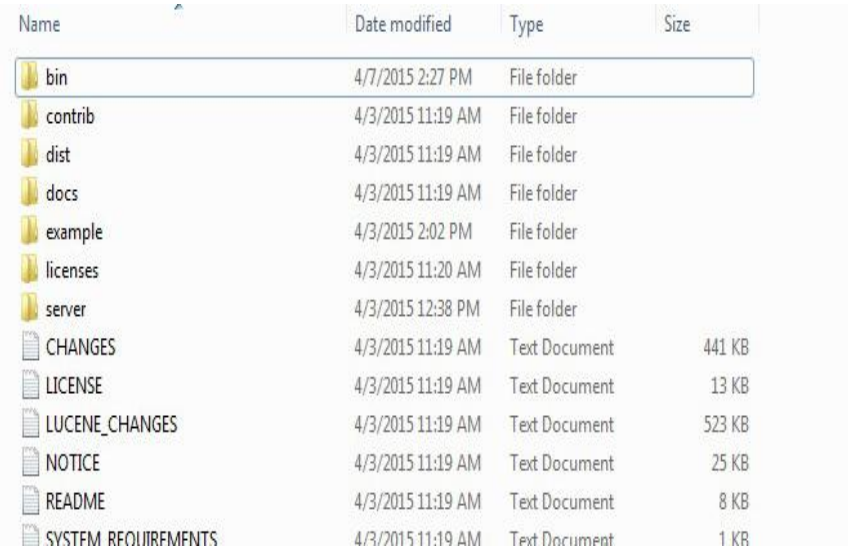
## 5. RESULTS



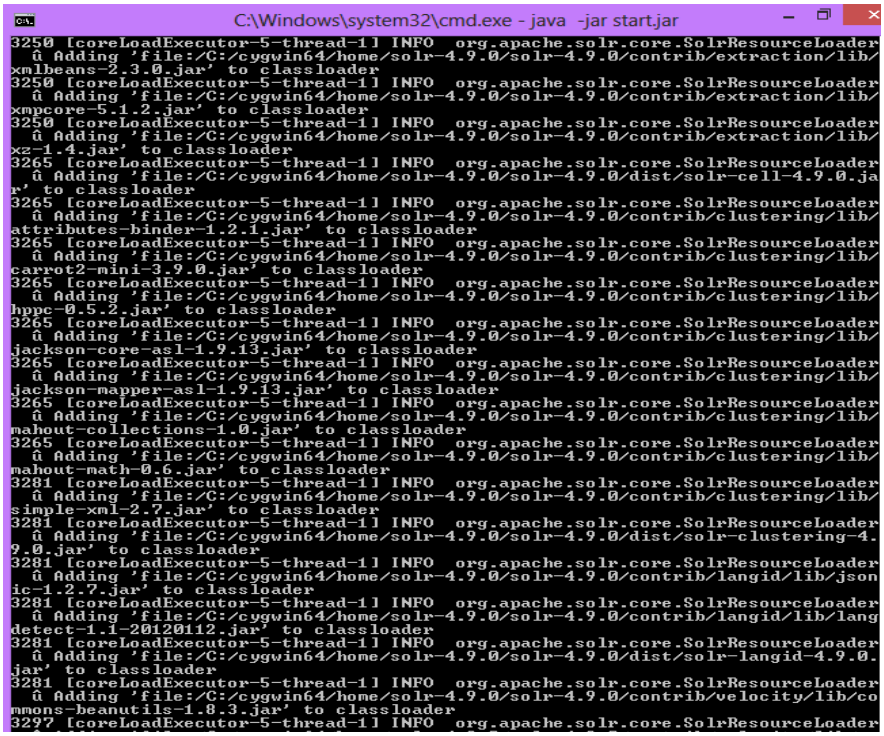

**Result 1.***Cygwin for Unix environment in windows OS to check whether the required files or folders are ceated or not. Cygwin command prompt for creating nutch directory*



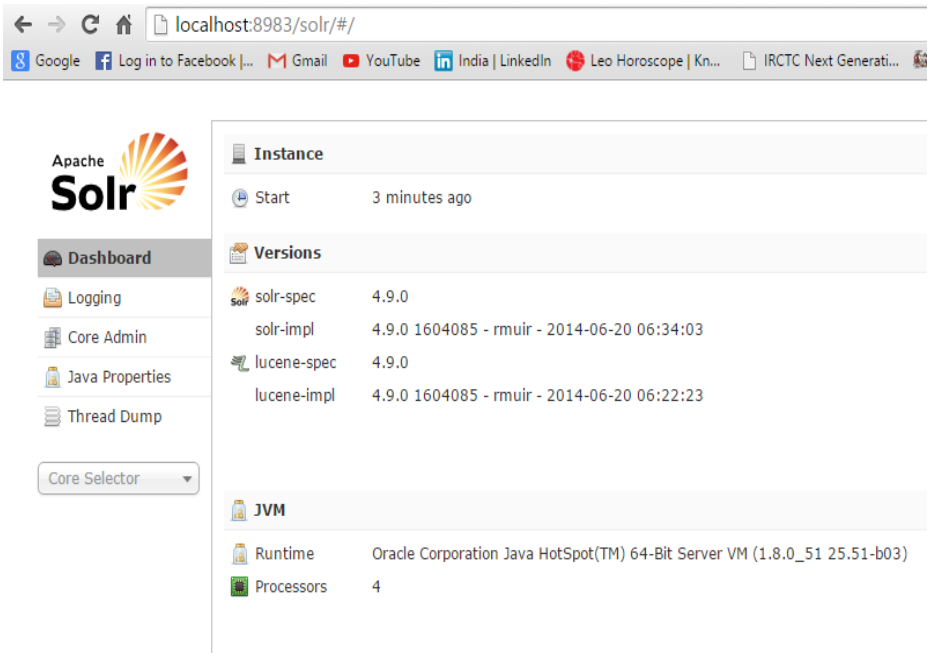

**Result 2.***Creating Nutch Files*



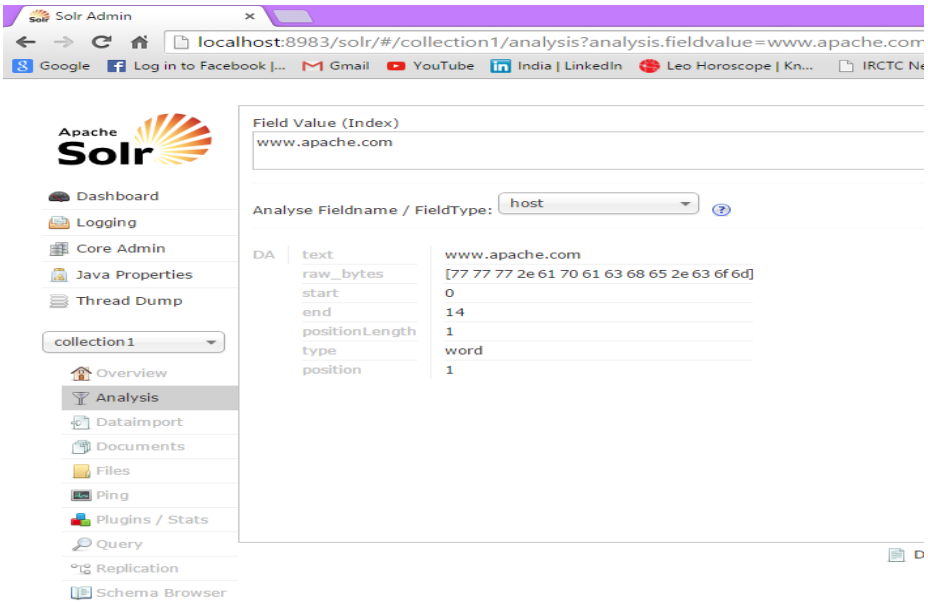

**Result 3.***Solr Folder*



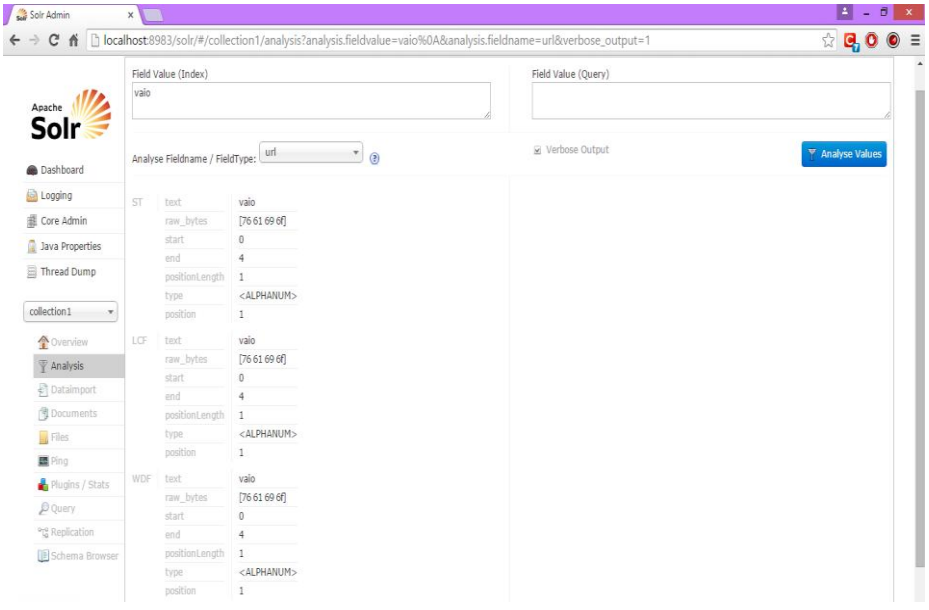

**Result 4.***Starting the Solr Server*



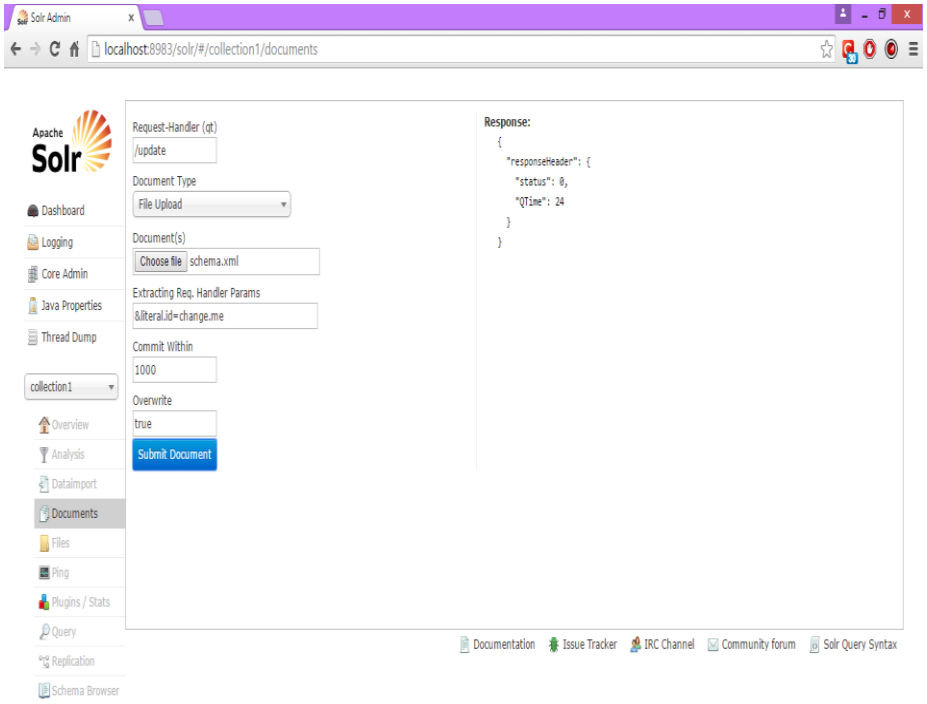

**Result 5.***Solr Admin Page*




**Result 6.***The analysis of the Apache webpage based on its host value.*




**Result 7.***Data retrieval based on its url*




**Result 8.***The response given to the uploaded document schema.xml*

## 6. CONCLUSION

We have described the architecture and implementation details of our crawling system, and presented some preliminary experiments. There are obviously many improvements to the system that can be made. A major open issue for future work is a detailed study of the scalability of the system and the behaviour of its components. Since Natural Language Processing techniques like text-search, placing the links according to their order and then connecting them to the dashboard and the output is visible in different forms like pie-diagrams, bar charts, and other forms etc. Here also language compatibility is discussed as it can be changed to any human language by its Unicode and it is done easily by the user individually without any permissions as the entire project is an Open Source.

## REFERENCES

[1] Ankita Dangre et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (1) , 2014, 921-922.

[2] Miguel Angel Rios Gaona, Alexander Gelbukh, SivajiBandyopadhyay ,Web-based Variant of the Lesk Approach to Word Sense Disambiguation, IEEE 2009..

[3] Kenneth W. Church and Lisa F. Rau, Commercial Application of Natural Language Processing, ACM 0002- 0782/95/1100

[4] Charu C. Aggarwal ,Fatima Al-Garawi, Philip S. Yu], Intelligent Crawling on the World Wide Web with Arbitrary Predicates, WWW10 May 1-5, 2001, Hong Kong. ACM 1-58113-348-0/01/0005.