Towards a Distributed P2P Link Analysis Based Search System

Name: Bhuvan Bamba  
GTID: 902092376

CS8803 Advanced Internet Application Development Project Report
Abstract

Search engines provide a reliable and easy way of looking for information on the World Wide Web. Users can express their query needs in the form of a keyword search. This keyword search results in a set of output documents which form the result of the keyword search. The keyword search interface provides a simple and reliable means of extracting relevant documents among the billions of pages available on the World Wide Web. The functionality of a search engine is shrouded in mystery due to lack of literature on the construction of a successful search system. In this project we build a search system and document our experiences with building this system.

We use the distributed approach to building a search system using a Peer-to-Peer System. The system is currently deployed on a single machine but is easily extendible to multiple machines to form a distributed network of machines which constantly crawl and index documents available on the World Wide Web. The system provides a graphical user interface (GUI) which allows the user to execute each component of the system with the click of a single button.
1. Introduction

A search system comprises of three different components - crawler, indexer, search and ranking component - each of which is essential for efficient performance of the system. Each of these three components involves functions which are time and memory consuming. A crawler constantly downloads documents from various servers hosting documents in the World Wide Web. These documents are cached on disk and passed onto the indexer. The indexer parses the documents extracting relevant keywords from the documents and forming a forward index. This forward index is basically an association between documents and the keywords occurring in these documents. This index is not suitable for conducting keyword searches. It needs to be inverted, which involves a reverse association where the system maintains a list of keywords and relevant documents associated with each of these keywords. This is known as an inverted index, as it is formed by “inverting” the original index. This index can be searched for extracting relevant documents by executing a keyword search on the system. The user expresses her query needs as a set of keywords. A set of documents relevant to the keywords is output as the result. The set of relevant documents can be ranked using traditional information retrieval scoring functions like the TF-IDF measure. The system developed here uses Apache Lucene API for indexing and searching. The documents are scored using the TF-IDF measure and sorted in accordance with the relevance score obtained by each document.

The scores thus obtained make uses of the term frequency and inverse document frequency. Term frequency directly affects the score of relevant documents in accordance with the number of occurrences of relevant search terms in the documents. Inverse document frequency on the other hand gives higher scores to documents containing terms which occur rarely. A high weight in TF–IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tends to filter out common terms.

In the context of the World Wide Web, Link Based approaches are among the most popular ranking approaches. They make use of the inherent linkage based structure of World Wide Web documents assigning each document an importance score. This importance score is based on the incoming links for a document; a document which is pointed to by many high quality documents should have a higher importance score. Google’s highly popular search technology [1] exemplifies the success of the link based ranking algorithms in identifying important pages. However, such link analysis based algorithms suffer from some drawbacks. Google’s PageRank algorithm has an update time extending into months which is not feasible for frequent updating of the system. Secondly, the algorithm is susceptible to manipulation by malicious Web Spammers, who manipulate the link based analysis to favor their fake websites. The problem commonly termed as Web Spam can seriously hurt the performance of PageRank algorithm, leading the algorithm into providing unjustified high PageRank to spam web pages. In [2], the authors propose the SourceRank approach for enhancing PageRank through source-based link analysis, which can potentially help combat the problem of web spam.

As mentioned earlier each of the components of a search system has heavy memory requirements. Even the commercially available crawlers can crawl only a few billion pages. It
is beyond the capacity of these systems to discover the entire World Wide Web. Indexing, if limited to a few machines, can be a time consuming operation as observed in our experiments. For search and ranking, traditional information retrieval based measures don’t work particularly well in the context of the World Wide Web. The link based analysis methods are an alternative, but these methods are computationally very expensive and have an update time running into months.

In this project, we propose a distributed peer-to-peer based approach for overcoming the problems associated with the various functionalities of a search system. The system uses the Apoidea[3] Peer-to-Peer crawler developed at Georgia Institute of Technology. This crawler is implemented using the chord routing protocol, hence it inherently implements a structured Peer-to-Peer system for implementing the crawler. The indexing and search components are implemented using an open source API, Apache Lucene. The rest of this report is organized as follows. Section 2 provides an overview of the proposed approach for implementing a peer-to-peer search system. Section 3 provides a description of the web based application implemented and deployed on a machine. The implementation details for the system are provided here. The crawler, indexer and search components are described in the following sections. This is followed by a brief summarization of the contributions made by this project in section 7. Section 8 provides a glimpse of the proposed future work. This is followed by the conclusions in section 9.

2. The Big Picture

This section provides an overview and discussion of our proposed approach for building a Peer-to-Peer search system. We discuss the pros and cons of building a system using the structured peer-to-peer approach and the unstructured peer-to-peer approach to analyzed which approach will work better for a search system.

2.1 Peer-to-Peer Search System: The Big Picture

As mentioned earlier, the current implementation of this project does not support a distributed peer-to-peer based search system implementation. However, we provide the ideas associated with implementation of such a system.

As seen in Figure 1 below, the system is a distributed Peer-to-Peer system in which each peer is responsible for crawling one or more specific domains. This is termed as source specific crawling. Source specific crawling is based on the ideas advocated by the source rank approach developed by the authors in [2]. Each peer can crawl one or more domains and any URLs belonging to other domains encountered by this peer are sent to the peer responsible for crawling that domain. Section 4 describes this approach as implemented in Apoidea in some detail.

As far as the indexing is concerned, each peer can handle the indexing of the domains it has crawled. This index will be used for searching the collections or sources crawled by this peer. Search functionality is implemented and utilized by each peer for searching the document sources (collections). This requires a distributed full text keyword search, where a peer can
execute a keyword search on its own index and at the same time route this search command to other peers each of which will return relevant results to this peer. Search functionality in unstructured P2P systems is well studied and we exclude any further discussions related to this topic. The keyword search is required to support the link analysis based source rank implementation mentioned earlier. Source rank proposes to treat a collection of pages (termed as a source) as a single entity in the web graph used for link based analysis. The impact of this approach is that the web graph comprising of billions of pages is reduced to a web graph comprising of millions of sources or collection of pages. Computation of page rank on this reduced web graph is orders of magnitude faster than the traditional Page Rank based computation. This effectively reduces the time for updating the rank of pages. Each page can be given a source rank as well as a local page rank. The computation of the local page rank is limited to the web graph comprising of a single source.

![Source: www.ibm.com](source.png)

For example, in the figure peer P1 will calculate the page rank for each page in the www.cc.gatech.edu domain. A source rank for the www.cc.gatech.edu domain will be calculated by accumulating the information for all sources from the various peers. This can be done by providing all the source based links to a single peer. This peer will be responsible for maintaining the source based web graph and calculating the source rank for all the sources from this graph. Peers on the Internet have different capabilities. A more powerful peer can be selected for maintaining this source based web graph.
2.2 Structured vs. Unstructured P2P Systems: The Big Picture

In this section, we consider the options for building the Peer-to-Peer system. P2P systems can be structured or unstructured. Now we discuss which type of P2P system will constitute a better choice for building our search system.

As far as the crawling functionality is concerned, a structured P2P system seems to perform better than the unstructured P2P systems. The crawling component requires that duplicate pages are not downloaded. This requires that all peers communicate with a central server in order to distribute crawling jobs. A structured P2P system can use the chord protocol to distribute jobs. The Apoidea [3] crawler uses a chord based routing protocol to distribute jobs. Details related to division of labor in Apoidea are provided in a later section. An unstructured P2P system does not have any protocol for supporting this division of labor. As each peer is aware of only a limited number of peers in the network it is unable to uniformly distribute domains for crawling. Any domain may be crawled multiple times in such an unstructured P2P system. Structured P2P systems avoid this duplication by maintaining bloom filters while unstructured P2P systems can’t avoid duplication completely even if they use a bloom filter. On the other hand, duplication in P2P systems is an essential feature as replication of data makes data more easily available. Extensive literature exists for replication in P2P systems, which we do not discuss here. In conclusion, duplication of web pages might not prove to be such a bad idea!

The indexing component for the system would require the index to be created separately on each peer. Each peer crawls a domain (or set of domains) and indexes the documents that it has crawled.

The search and ranking component of the system will be better supported by an unstructured P2P system. Full text search for P2P systems is an extensively studied topic. For structured P2P systems well known search mechanisms are deterministic. Given a file identifier, the system is able to locate if the file exists in the system. No specific mechanisms exist for a keyword search in structured P2P systems. An unstructured P2P system has definite advantages as far as the search component of the system is concerned. Link Based Source Rank can be handled by each peer for the documents it crawls.

This suggests that a hybrid of structured and unstructured P2P systems might be the best option for building a P2P based search system.

3. System Implementation

The system that is currently implemented does not support a peer-to-peer implementation, although the Apoidea crawler is a structured P2P system. We have implemented a complete search system. The search system implementation is a web-based implementation. It provides a graphical user interface for executing the various functions associated with a search engine. The application is developed on the Apache tomcat 5.5 Web Server [6]. The crawler used in our implementation is a modified Apoidea based crawler. We made changes to the Apoidea [3] crawler to experiment with the www.cc.gatech.edu domain. This involved making changes to the rule based crawling functionality of Apoidea. Further discussion on
Apoidea is provided in a later section. Apoidea is a structured P2P crawler and inherently supports the P2P based system we propose to develop.
Figure 3 above displays the user interface for the Apoidea crawler. The user can enter the starting URL (seed URL) and seed Domain and download all documents in that domain to a specific folder. The seed Domain limits the peer to download documents from this particular domain only and prevents it from downloading documents belonging to other domains.

The indexer and search functionality is implemented using the Apache Lucene [7] API. Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is suitable for applications that requires full-text search, especially cross-platform applications. The indexer and search screens are displayed in Figure 4 and Figure 5, respectively. The indexer accepts an input folder, which is the directory to be indexed, and an index location. The index is formed in this index location. The search interface accepts keywords from the user. More details on the indexer and search interfaces follow in section 5 and section 6.
4. Crawler

This section provides a brief overview of the Apoidea crawler. The description of the Apoidea crawler is as mentioned in [3]. This is followed by a brief description of changes implemented on the crawler to perform a source based crawl. A few problems with the crawler were revealed by analyzing the www.cc.gatech.edu domain. We also provide the analysis and experimentation performed using the Apoidea crawler.

4.1 Apoidea: Decentralized P2P architecture for WWW crawl

Apoidea consists of a number of peers all across the WWW, where each peer is responsible for a certain portion of the web. In this section we describe how Apoidea performs the division of labor and how the duplicate checking is done. These are the two most important features of the Apoidea architecture. For other details related to Apoidea system, interested readers are referred to [3].

Division of Labor: Apoidea use a DHT-based system for distributing the WWW space among all peers in the network. A Peer P is responsible for all URLs whose domain name hashes onto it. In Apoidea, this is done using the contiguous region policy, which will be clearer from the example shown in Figure 6. Assume that there are three peers in the system - Peer A, B and C. Their IPs are hashed onto an \( m \) bit space and hence, will be located at three points in the modulo \( m \) ring. Then, various domain names are also hashed onto this space and they will also occupy such slots. The Peer A is made responsible for the space between Peer C and itself. Peer B is responsible for space between Peer A and itself and Peer
C with the rest. From the figure, Peer A is responsible for the domain www.cc.gatech.edu. Therefore Peer A will be the only peer crawling all the URLs in that domain. If any other peer gets URLs belonging to that domain, it batches them and periodically sends them to the Peer.

However, such a random assignment of URLs to peers does not exploit the geographical proximity information. In order to ensure that a domain is crawled by a peer close to it, the tight mapping of a domain to a peer can be relaxed. Our system does not intend to use this feature of Apoidea at the moment and we skip any further discussion associated with this relaxed mapping.

**Duplicate Checking:** Duplicate checking is a very important feature of the Apoidea system as it prevents the system from downloading duplicate pages. The extent of duplication on the WWW requires that this particular feature be implemented. The benefit of this feature would be in form of time saved by prevention of download of duplicate pages. Let us assume that Peer P is responsible for domain D. When any peer encounters a URL in domain D, it sends the URL to Peer P since it is responsible for domain D. Now, Peer P maintains a bloom filter which indicates as to whether a URL is already crawled or not. Hence, Peer P checks if a given URL is already crawled or not. If the URL were indeed already crawled then it is simply dropped, else it is added to the Crawl-Jobs list.

In order to check page duplication, page contents are hashed onto the identifier space and distributed amongst the various peers in the network. Figure 6(b) shows an example of it. The page content of www.cc.gatech.edu/research is hashed onto the ring and Peer C is responsible for it. Note that the page must have been crawled by Peer A because the domain www.cc.gatech.edu hashed onto it. When Peer A encounters that page, it needs to check whether that page is a duplicate or not. So it looks up for the hash of page content and finds that Peer C is responsible for it. Now Peer A can batch all such requests and periodically query Peer C about the newness. Peer C can then batch the replies back. It also modifies its local Seen-Content list to note the new pages that have been downloaded by Peer A. Note that the delay in getting the information about the duplicity of a page does not affect
performance, as there is always a significant delay in downloading a particular page and processing that page. This happens because downloads typically happen at a much faster rate than the processing and as a result the processing has a significant lag from the download.

4.2 Crawler: Analysis

We carried out source specific crawls for crawling the www.cc.gatech.edu domain using the Apoidea crawler. Our repeated analysis of the crawls for this domain revealed problems with the original Apoidea crawler. The current Apoidea crawler implemented in Java language makes use of The Java Regular Expressions package to extract URLs from a crawled page. We search for patterns of the form `<|\s*a\s+href\s*=\s*"?\s*s*=` in the HTML text. The above pattern is quite simple even though it looks complicated! It simply goes through the text while looking for patterns of the form “<a href=” along with its common variations. We all know that this is the tag used to refer to an external HTML page (or another file) from a particular page. This particular tag forms the basis for the random walk model used by Google’s PageRank and many other Link Analysis Based algorithms. We analyze the performance of the Apoidea crawler here in terms of the quality of its crawl. Does the Apoidea crawler work as expected? Does it extract all relevant URLs from the web pages? Is it missing any URLs? These are a few of the questions we explore in this section.

Our familiarity with the given domain (www.cc.gatech.edu) was a crucial factor in the successful analysis of the crawler. It is a very difficult task to point out if the crawler is missing any pages. The original Apoidea crawler had some incorrect rules which prevented certain URLs from being crawled. For example, relative URLs starting with a “/” were being excluded. There does not seem to be any reason to prevent such relative URLs from being downloaded. We fix these incorrect syntax rules in Apoidea thus ensuring that all URLs are extracted correctly by the crawler. In the next section, we point out a specific example where the current Apoidea regular expression based rules are not sufficient for extracting all URLs.

4.3 Crawler: A Specific Example

In this section we analyze the web page given in Figure 7. This particular webpage has links to student web pages and these links in particular were a noticeable omission from the crawl of the www.cc.gatech.edu domain. We downloaded this web page and attempted to analyze why the links to the student web pages were not being extracted. We discovered the following: there are different ways of referring to a particular web page. In our example the crawler is looking to extract all URLs embedded in the “<a href=” tag. This particular web page as shown in Figure 8 below is referred to in a different way, using a “src” tag. The links to all student pages are on a separate web page, namely http://www-static.cc.gatech.edu/people/index-cms.grad.html. This web page is referred to by including it inside a “src” tag instead of an “<a href=” tag. As our crawler is not looking for this particular pattern, we miss out on these web pages. Our knowledge of this particular domain helped us point out that the links to student web pages were not being extracted, which in turn helped us correct this flaw with the Apoidea crawler. Now we include all references to the “src” tag in our Regular Expressions to extract URLs inside the “src” tag. In the next section we provide our results related to the crawl of the www.cc.gatech.edu domain.
Figure 7 Specific Links not being extracted by Apoidea

Figure 8 HTML for the web page indicating that the URL is referred to inside a "src" tag.

This option will not work correctly. Unfortunately, your browser does not support inline frames.
4.4 Crawler: Experiments

We carried out an experimental analysis of the Apoidea crawler attempting to improve its performance in terms of the number of URLs crawled. We give our results in Figure 9 below. *Run 1* is the performance of the original Apoidea crawler which manages to download only 1812 web pages. As mentioned in an earlier section, this version of Apoidea was suffering from a few problems including the incorrect exclusion of relative URLs which badly hurts the performance of the crawler. *Run M* downloads nearly 18,000 web pages. This is the performance of the crawler after fixing the incorrect rules in the original Apoidea crawler. We do not have any specific idea as to how many times the crawler was tested before we accepted this as being the performance of the crawler, hence it is tagged as *Run M*.

*Run N* is tagged so because of similar reasons and it indicates the performance of our crawler after we include all URLs being referred to by the “*src*” tag. During Run N we are able to crawl as many as 25,000 web pages. Some doubts arise regarding the correctness of these results when these documents are indexed and a search is conducted. As we mention in a later section, we notice that duplicate documents are being downloaded. This brings us to the conclusion that we should be familiar with the HTML syntax rules to ensure that we are extracting all relevant URLs. In our experimentation our familiarity with the particular domain proved to be helpful in analyzing the performance of the crawler. When we are attempting to crawl the entire web any such analysis is entirely unfeasible due to the extent of the World Wide Web. The W3C syntax rules for Markup languages [8] should be well studied and followed while writing a crawler.

![Figure 9 Crawler performance on www.cc.gatech.edu domain](image-url)
5. Indexer

The indexer component of our search system is implemented using the Apache Lucene API. We implemented a web based interface for the indexer component as shown in Figure 4. The indexer is a *scalable full text indexer* which forms an *inverted index*. An *inverted index* is an index structure storing a mapping from words to their locations in a document or a set of documents, giving full text search. An inverted index is the most important data structure used in search engines. Such an associative array is a multimap, and can be implemented in many ways. It could be a hash table, where the keys are words (strings), and the values are arrays of locations.

There are two main variants of inverted indexes: An *inverted file index* contains for each word a list of references to all the documents in which it occurs. A *full inverted index* additionally contains information about where in the documents the words appear. This could be implemented in several ways. The simplest is perhaps a list of all pairs of document IDs and local positions. An *inverted file index* needs less space, but also has less functionality. You can do *term search* (what you usually do in a search engine), but not *phrase search* (what you usually get when you put quotes around your search query). Apache Lucene implements a full inverted index providing additional features such as incremental indexing. This means that more documents can be added to an existing index which is supported by the *Add to Index* radio button as shown in Figure 4. Indexing is a lengthy process and our implementation gives an idea to the user about the progress of the indexer. A progress bar appears once the user submits an indexing job through the web based application(Figure 10). Once the indexing is complete the user can view the statistics associated with the indexing job(Figure 11).

![Figure 10 Indexing Progress Bar](image-url)
The size of the index is approximately 23% the size of the indexed text. Our experiments reveal that the indexer indexes around 12MB of text or 1600 documents per minute. This assumes an average document size of 8KB. At this rate to index a set of around 8 billion web pages, we require 9.5 years to index the entire collection of pages on the single peer on which the system is implemented! This shows that the indexing functionality should be supported by a number of peers and must not be limited to a few machines.

6. Search

The search functionality implemented utilizes the Apache Lucene API in conjunction with the indexer functionality mentioned in the section above. The system currently supports ranked searches based on the TF-IDF measure, which is a traditional measure used in Information Retrieval. Lucene supports powerful query types: phrase queries, wildcard queries, proximity queries, range queries are all supported by the system. Further the system supports field searching (e.g., title, author, contents) for XML docs, date-range searching, sorting by any field. Lucene provides for multiple-index searching with merged results which supports our goal of conducting distributed searches through a P2P system. Simultaneous update and searching over the index is permitted which ensures that the system will keep running at all times. The system is currently enabled for HTML/Text Search only and will be extended to include additional search functionality. The current implementation does not support any Link Based Analysis. This means that the documents are ranked in accordance with the TF-IDF measure only and do not employ any Page Rank based ranks. The web
based search is conducted from the text field as shown in Figure 5. Results are displayed on a web page with hyperlinks to the web pages as shown in Figure 12.

![Peer Search Interface](image)

Figure 12 Search Results displayed with hyperlinks to web pages

7. Contributions
This project involved building an entire P2P based search engine. The specific contributions made in this project were:

- **Working Web-based implementation of a complete search system (with some glitches!!):** We developed a web based application for the search system using an Apoidea based crawler and the Apache Lucene API for indexing and search functionality.

- **Improvements to Apoidea Crawler:** As discussed in section 4 we analyzed and experimented with the Apoidea crawler to improve its performance in terms of the quality of the crawl. We made some changes to the original Apoidea crawler to ensure that it extracts links correctly from web pages.

- **Possible framework for a P2P Web Search System:** Lastly, we proposed a possible framework for supporting a P2P Web Search System.

8. Future Work
Possible directions for future work include extending this to a P2P based system. As mentioned, the Apoidea based crawler already supports a structured P2P functionality based on the chord [9] protocol. We need to extend the indexing and search functionality to support
this P2P based framework. Secondly, a source rank based approach towards link analysis needs to be implemented on this system. Using this system, we hope to display the advantages of source rank over page rank.

9. Conclusions

In conclusion, we would like to state that a P2P based approach towards building a search system seems to be a very feasible option as we try to tackle the problem of searching for relevant information quickly on an ever expanding repository of data on the World Wide Web. The best approach seems to be a combination of structured and unstructured P2P systems as neither structured nor unstructured P2P systems alone supports all three components of a search system in an efficient manner. We have taken a first step towards building such a system by implementing all the components of a complete search system on a single machine.

References


[8] W3 Standards for Markup Languages http://www.w3.org/MarkUp/