SCALABILITY OF STEPPING STONES AND PATHWAYS

Logambigai Venkatachalam

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Thesis Committee:
Dr. Edward A. Fox, Chair
Dr. Weiguo (Patrick) Fan
Dr. Mohamed Kholief

April 21, 2008
Blacksburg, Virginia, USA

Keywords: Scalability, Lucene, CiteSeer,
Connection finding search framework

SCALABILITY OF STEPPING STONES AND PATHWAYS

Logambigai Venkatachalam

Committee Chairman: Dr. Edward A. Fox

(Abstract)

Information Retrieval (IR) plays a key role in serving large communities of users who are in need of relevant answers for their search queries. IR encompasses various search models to address different requirements and has introduced a variety of supporting tools to improve effectiveness and efficiency. 'Search' is the key focus of IR. The classic search methodology takes an input query, processes it, and returns the result as a ranked list of documents. However, this approach is not the most effective method to support the task of finding document associations (relationships between concepts or queries) both for direct or indirect relationships.

The Stepping Stones and Pathways (SSP) retrieval methodology supports retrieval of ranked chains of documents that support valid relationships between any two given concepts. SSP has many potential practical and research applications, which are in need of a tool to find connections between two concepts. The early SSP 'proof-of-concept' implementation could handle only 6000 documents. However, commercial search applications will have to deal with millions of documents. Hence, addressing the scalability limitation becomes extremely important in the current SSP implementation in order to overcome the limitations on handling large datasets.

Research on various commercial search applications and their scalability indicates that the Lucene search tool kit is widely used due to its support for scalability, performance, and extensibility features. Many web-based and desktop applications have used this search tool kit to great success, including Wikipedia search, job search sites, digital libraries, e-commerce sites, and the Eclipse Integrated Development Environment (IDE).

The goal of this research is to re-implement SSP in a scalable way, so that it can work for larger datasets and also can be deployed commercially. This work explains the approach adopted for re-implementation focusing on scalable indexing, searching components, new ways to process citations (references), a new approach for query expansion, document clustering, and document similarity calculation. The experiments performed to test the factors such as runtime and storage proved that the system can be scaled up to handle up to millions of documents.

Acknowledgements

First, I would like to express my sincere thanks to my advisor and thesis committee chairman Dr. Edward Fox for his guidance during this research work. He introduced me to the field of information processing and inspired me to work on this research. I am grateful to him for providing the intellectual space and for unwrapping the research interest in me.

I have to thank my other committee members, Dr. Patrick Fan and Dr. Mohamed Kholief, for their review comments and encouraging words all through this research. I would like to thank Al Cooper and Brock Burroughs from Business Management Systems, for providing an assistantship and funding my education throughout my Masters degree.

I would like to thank my friends and colleagues at the Digital Library Research Laboratory for their support and feedback throughout my time here. I express my gratitude to Xiaoyan Yu for her counsel. I would like to thank Ashwin Aji for his contributions in this project and his valuable inputs. Thanks go to Uma Murthy for encouraging me and endorsing my ideas, during my stint at the laboratory. I have to thank all my friends at Virginia Tech for making my stay a memorable one, and also friends back home who constantly encouraged me. There was never a dull moment, when I was in the company of Jai Shankar and Ajith Sowndararajan; I would like to thank them for making my stay pleasurable. Also, I thank my roommates Anusha, Sneha, and Tilo who constantly encouraged and backed me in all my pursuits.

My greatest thanks go out to my parents for inculcating values like industry, honesty, sincerity, and for instilling the adventure spirit to chase my dreams. My sincere thanks to my brother for his encouraging words, love, and support.

Table of contents

List of Figures

List of Tables

Chapter 1. Introduction

In the current Internet era, as part of day-to-day activities, web users seek information on various topics. Information Retrieval (IR) plays a significant role in addressing the user's search requirements by providing appropriate systems and tools. IR has to constantly introduce new solutions and systems to address varying user requirements. For example, the 'multi-query connection' [1] is a sea change from the contemporary 'single query search'. Other catalysts to the innovations in the IR field include scalability improvements, distributed search, parallel search, search performance improvements, and new visualization methods.

A primary focus in computing research would be to develop working models to prove any new concept or methodology. However, while designing such models researchers may not consider scalability and performance requirements. Accommodating scalability and performance requirements in such models is critical before they are deployed. Thus, Stepping Stones and Pathways project [1], which was initially developed as a 'proof of concept' framework, focused primarily on addressing the different query interpretation requirements. This novel search methodology has the potential to be used in research projects and commercial applications. However, due to the scalability constraint in the original Stepping Stones and Pathways framework, it is not suited for large scale commercial applications. The primary focus of this research is to address the scalability issue in SSP, which would make it suitable for large scale systems.

1.1 Background of Stepping Stones and Pathways

Current search systems interpret a user's information request as a single query, retrieving answers as a ranked set of documents, in the order of their relevance to the query. This has proved to be successful and the IR field has techniques and tools developed for this type of search, many commercially deployed. Some constraints in this framework include:

- Users interpret that there will be a one-to-one mapping between the query and the documents in the document space. This is not always true and hence, there may be failures to satisfy user needs.
- In the event of a long query, it is treated as describing a single idea, which may be erroneous.

A different methodology (see Figure 1) was proposed to address the above issues, to interpret the information request (user query) better and to present the results to the user as a connected document set (chain of documents). Also, a novel idea of allowing user feedback can be added in the new system, to allow the user to interact with the results of the search.

The Stepping Stones and Pathways [1] approach aims at providing an alternative interpretation of user queries. In this interpretation, a query represents two related, separable concepts. The objective of the query then is to retrieve one or more sequences of document chains that support a ranked list of relationships between the two concepts. The input to the IR system is still a set of words, but now represents two separate concepts.

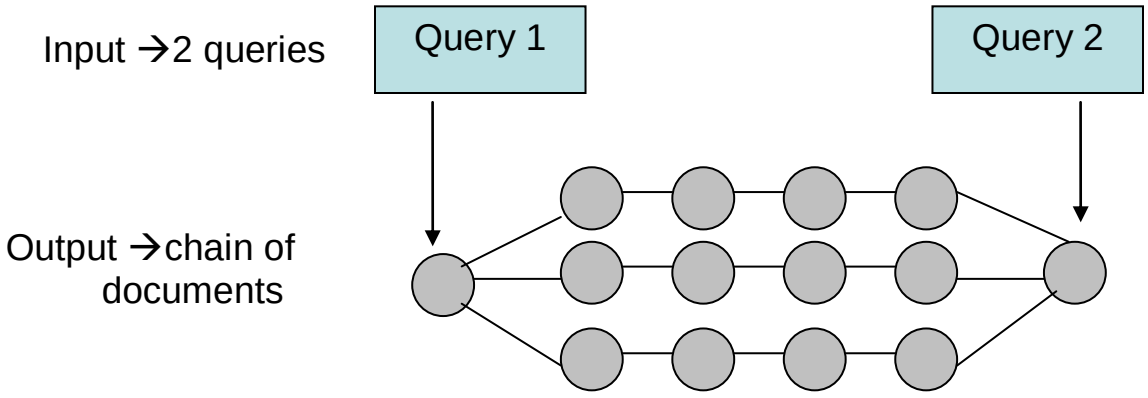

Figure 1: Stepping Stones and Pathways

Stepping Stones and Pathways (SSP) aims at handling hard queries, which are not covered by single documents. The solution is to find a group of documents that are

related and each of which covers a different aspect of the queries. This idea has many potential practical and research applications such as:

1. Defense organizations can use this approach to find relationships between various activities planned by attackers.
2. This can be used in the research project, "Mining evidences for relationships in heterogeneous object spaces", where the focus is to find the document connections.
3. Medical practitioners can use this approach to find relationships among various evolving research topics and discover new relationships between various concepts.
4. Any field that needs to discover relationships among various research ideas, events, people, etc. can make use of this application.

In essence the goal of SSP is to give users a framework, which can be used to find possible connections between concepts. This could help them discover interesting relationships. However to achieve this using SSP, the current framework should be modified to work with a larger data set. This problem is addressed in this research work.

1.2 Search Scalability

Search systems must be scalable, i.e., they must have the capability to handle a growing amount of data and processing seamlessly. The factors that determine scalability depend on the system under consideration. In the search field these factors include the number of queries handled per second, search time measured in milliseconds, number of gigabytes of data handled, and maintenance requirements. The system must be capable of supporting near real-time searchability of data and should scale linearly with the addition of any number of new documents and the number of users interacting with the system. Key components, which will assist in addressing the scalability problem, are developments in search system architecture, a wide range of parallelization techniques, partitioning search corpus data by allocation of tasks to multiple back end servers, and upgrading of the system configuration.

1.3 Motivation

Stepping Stones and Pathways (SSP) has great potential in commercial and research applications to find relationships between concepts. Currently, SSP can process only 6000 documents. In order to use SSP to handle a larger dataset, the serious limitation of scalability needs to be addressed.

While researching about various commercial search applications and the scalability of various existing search engines in general, we found that a search tool kit, Lucene, offers rich indexing and searching capabilities along with compelling features like analyzers and parameterization. Such features make Lucene a reliable, flexible, and robust framework. Many current search applications that are both web applications and desktop applications use this search toolkit and have proved to be a great success.

The key fact is that all these applications deal with large data sets and Lucene has handled those very effectively and efficiently. Lucene as a search framework is notable for its indexing and query processing capabilities. This is the motivation to use Lucene for the SSP application. This could help achieve the goal of making SSP work for larger data sets (Scalable SSP Framework).

1.4 Research Question

Stepping Stones and Pathways is used to find relationships between concepts and present results as a chain of documents. The current framework can handle only 6000 documents. With this background, the research questions, which are addressed in this work, are:

- How to address the scalability issue in Stepping Stones and Pathways?
- What is the size of the data set, which the new framework should be able to process in order to meet the scalability requirement?
- What applications benefit from this new scalable framework?

1.5 Hypothesis

The Stepping Stones and Pathways framework has scalability limitations and only can handle documents numbering in the thousands. This is because the current system uses an in-memory search engine, which introduces a smaller index size constraint.

We hypothesize that it is feasible to improve the scalability of the existing SSP framework, such that it can handle much bigger collections and can be put into practical use. To achieve this, a switch to an out-of-memory search engine like the Lucene search toolkit is necessary. Lucene is currently used by many search applications as their underlying search framework.

In this approach, Lucene is embedded into the original SSP framework to implement the indexing and querying functionality. Also, some of the other relevant tools for query expansion, matrix construction (with references), and graph construction are replaced. However, other steps of the SSP engine remain the same. The advantages of this framework include:

- Support for a much bigger collection (tested with approximately 700,000 documents)
- Handle data with size on the order of 100's of gigabytes

1.6 Claim and Evidence

Following are the claims to support the scalable SSP hypothesis:

1. The success of many existing search applications which are built using Lucene demonstrates that SSP could be made to work on a larger collection if it is modified to use Lucene as the underlying search tool kit.
2. From an implementation language perspective, Java is suitable for scalability because of its support for effective resource management, particularly memory management. However, there is a performance trade off when using Java and hence, appropriate measures are taken to improve performance.
3. Search applications based on Lucene are easy to develop. This reduces the development time for any new search application.

4. Lucene's indexing approach, particularly the segment merging approach, scales well and offer developers a high degree of flexibility in providing a trade-off between indexing and searching speed.
5. Lucene uses flat files as opposed to complicated Btrees.

1.7 Research Methodology

The research work is divided into many phases. It is presented in Figure 2.

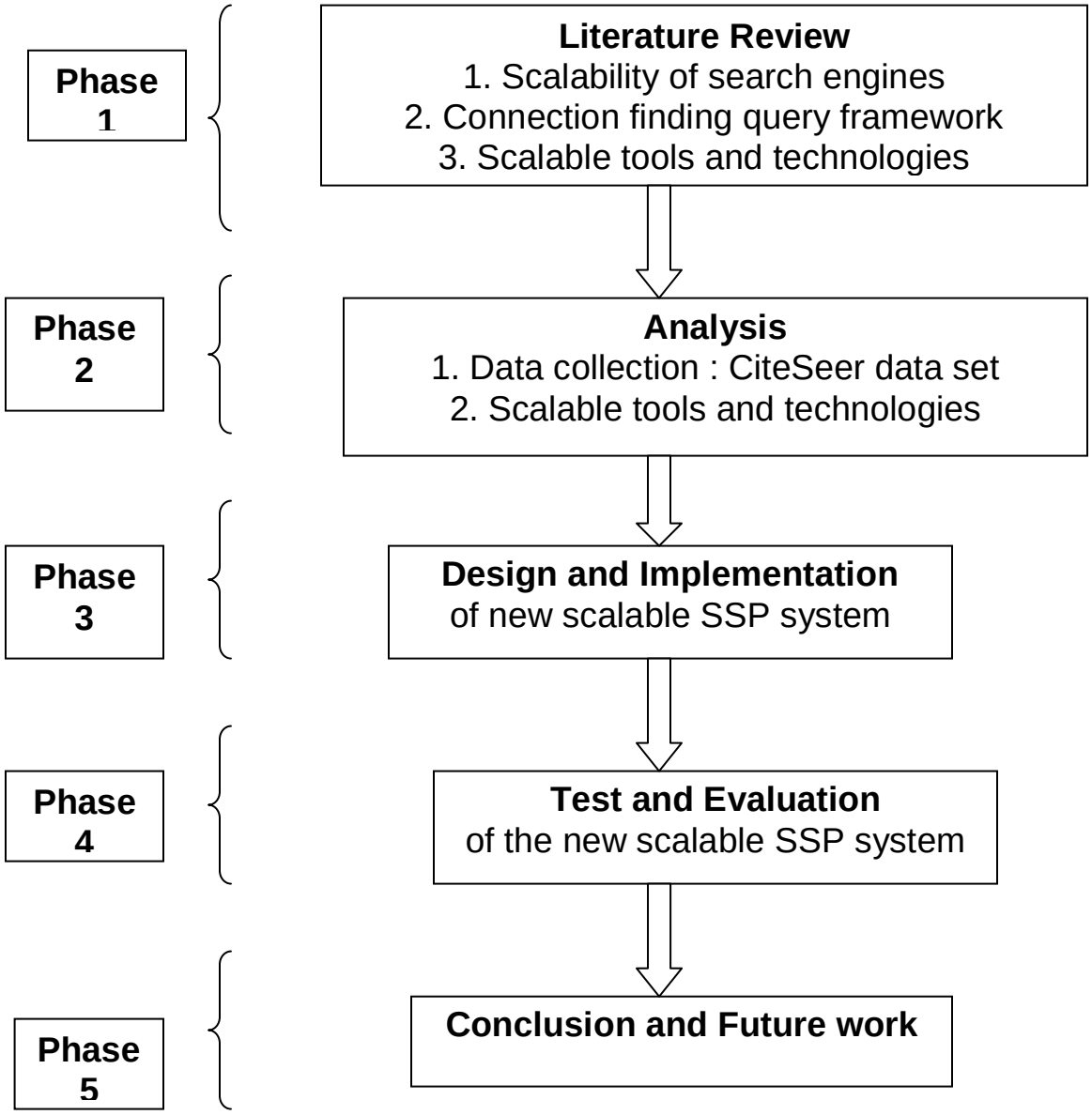

Figure 2: Research Methodology in Scalable SSP Implementation

Chapter 2. Literature Review

2.1 Summary

This chapter presents information about connection finding retrieval methodologies in contrast with SSP, scalability in information retrieval and related fields, and tools and technologies used in existing search systems that meet the scalability requirements. The Lucene search tool kit is also described in this chapter.

The following table shows the categorization of various works related to IR and the corresponding research work, which are referenced in those areas:

IR work categories	Research work referenced
Connection finding query frameworks	[2],[3],[4],[5]
Digital Library scalability	[6]
Scalability tools	[7, 8]
Web search and scalability	[9, 10]
Lucene based scalability	[11-13]
Others	[14],[15, 16]

Table 1: Relevant Research in Search and Scalability

2.2 Connection Finding Query Frameworks

The novel idea of a computer assisted approach to find connections between mutually isolated literatures was introduced by Swanson in the Arrowsmith system [17]. Research in the field of information processing, particularly search, has developed to a great extent. '*Finding document connections'* is one of the promising areas, which has huge research potential. To find connections between concepts, Deept et al. have developed an automated approach for biological knowledge discovery in their work "*Finding combinatorial connections between concepts in Biomedical Literature*" [2] where they find connections between biological concepts. Using these connections scientists can

subsequently perform actual experiments to verify and confirm these connections, which are initially automatically discovered by using the framework. SSP is similar to this work since SSP also aims at finding connections between concepts. However, the document similarity calculation strategy, features supported, and data set used are different in both systems.

Gresock et al. in their work [4] have developed a framework to find connections between concepts using PubMed documents (biomedical publications). This project involves processing nearly 500,000 documents, the data set size that is close to the data set size requirement of the new scalable SSP implementation. Storytelling work is different from SSP in the following ways:

1. In Storytelling the following steps are performed as off line tasks:
    a. Filtering.
    b. Document-Document similarity calculations.

   SSP performs this during runtime.
2. Storytelling uses a database for storing indexed information as opposed to the index structure used in SSP.
3. Storytelling focuses on building static stories.
4. Users cannot change/interact with the output presented to them in Storytelling. However, in SSP user interaction is allowed. The end users can dig into many levels to understand the documents behind the connections. This offers great flexibility.

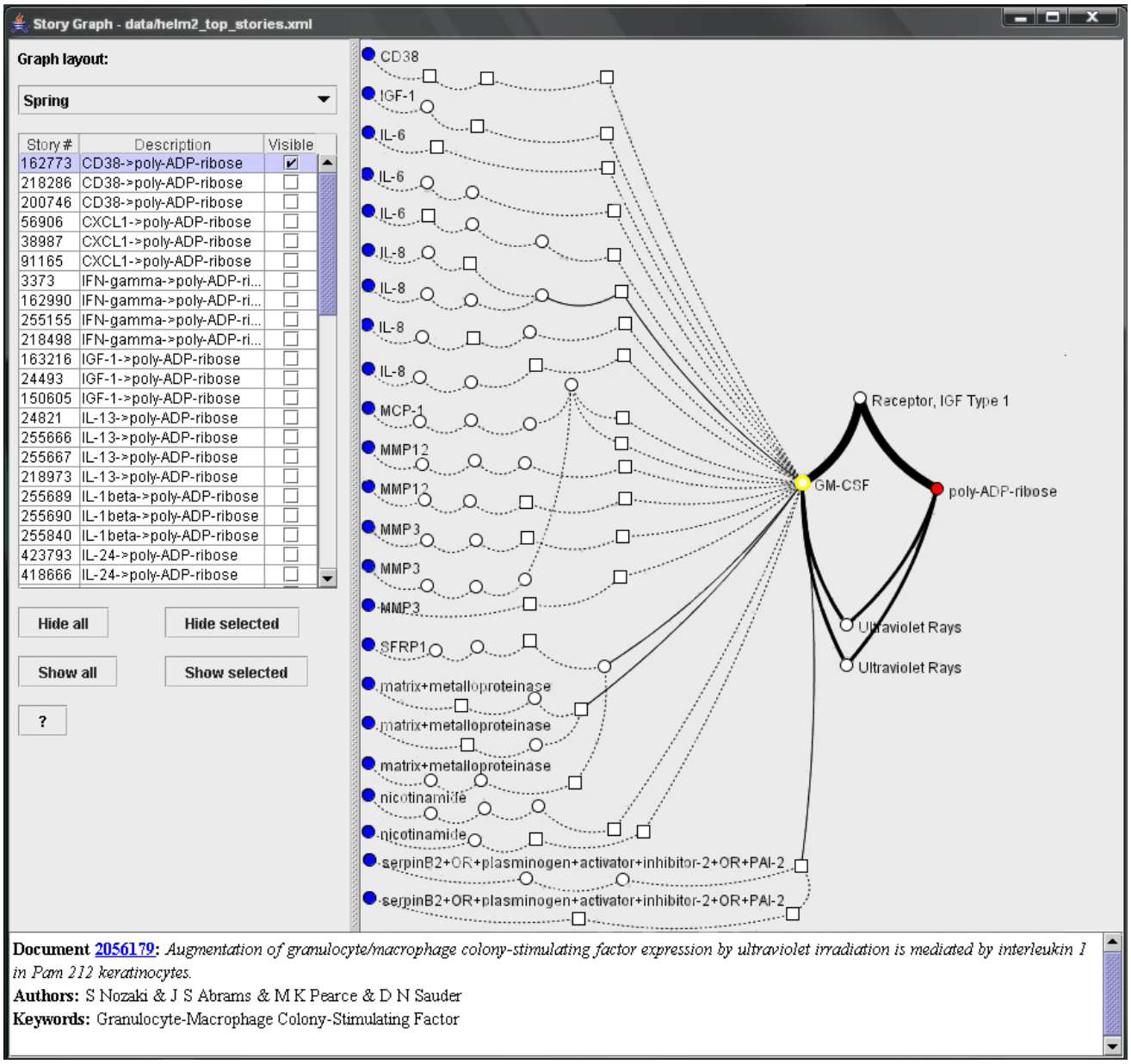

Figure 3: Storytelling (Reference [4], used with permission of Joseph Gresock)

Wikipedia, the widely used information site, is a rich data source to find associations, since it contains information from various domains. Omnipelagos is a connection finding query framework that is built to query the Wikipedia data source. It allows users to explore connections among various entities and events and presents the shortest path between them in the form of an ordered group of document lists. This framework is designed to work for concepts like people, places, things, events, and concepts. Each

chain is displayed along with a short text fragment, which provides context. It can be used for fact-finding, brainstorming, or just plain fun. It is a commercially deployed search application and is available at **http://www.omnipelagos.com/**

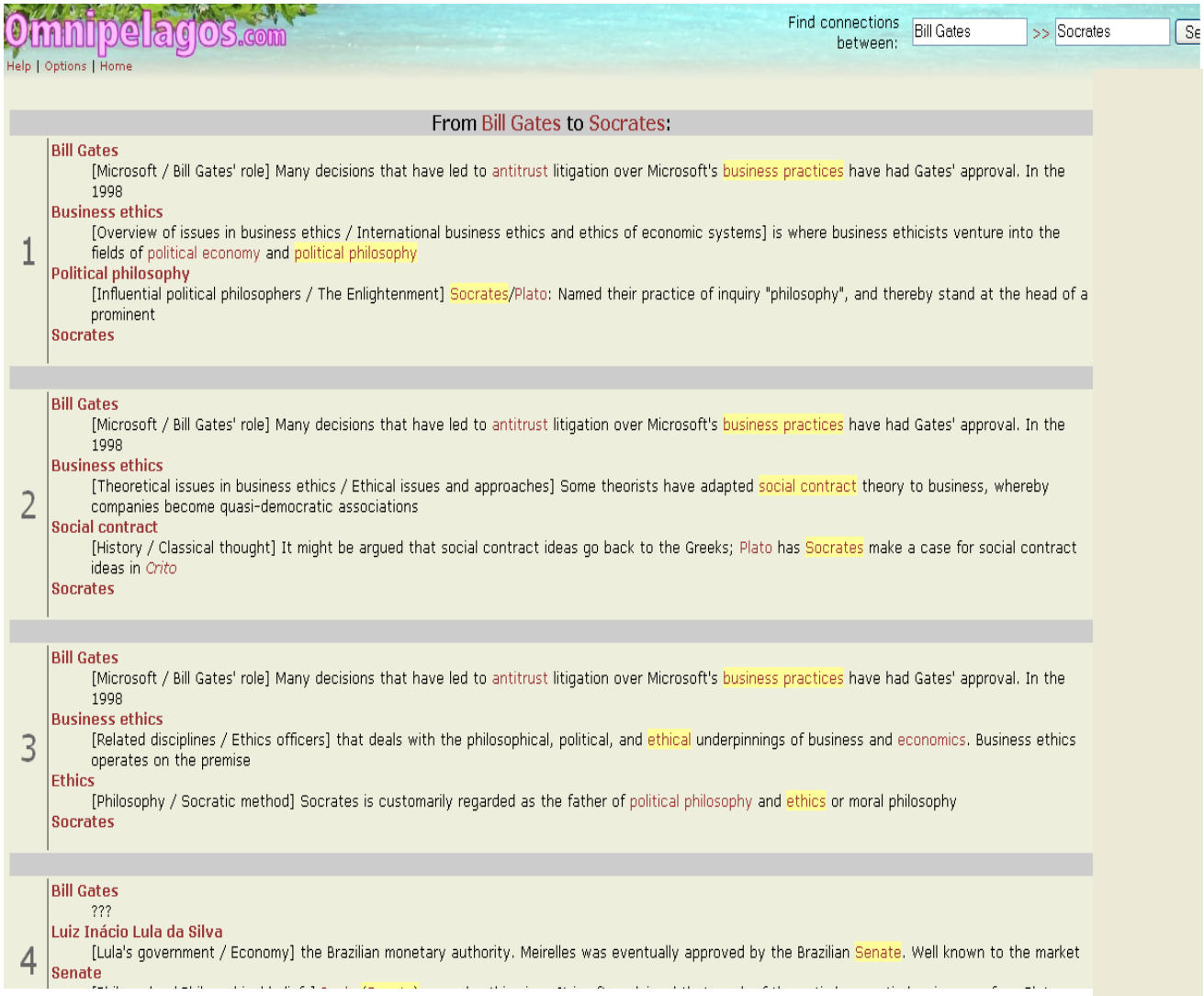

Figure 4: Omnipelagos (Source : www.omnipelagos.com, May 29, 2008)

SSP and Omnipelagos differ in the following ways:

1. Omnipelagos focuses on entity and event data whereas SSP focuses on scientific papers.

2. Omnipelagos presents the result, i.e., the set of documents connecting two concepts, as an ordered group of connection document sets whereas SSP shows the connections in two separate parts:
    i) The top part of the user interface (UI) shows connection documents as graphs.
    ii) The lower part of the UI shows a table of connection documents.
3. Omnipelagos provides search customization options like:
    a. Maximum number of results returned.
    b. Search strategy (longer or shorter query).
    c. Exclusion of topics from search.

    However, in SSP there is an option for user interaction to dig into many levels to see the lower level of connecting documents.

Dean et al. [14] deals with a different type of search input, i.e., URL of the page instead of query terms, and the output is a set of related pages. In contrast SSP takes as input two different queries (related or transitive) and outputs chains of documents connecting these concepts.

Johnson has used the Lucene search library to discover relationships between Gene Ontology and three other Ontologies. They evaluated lexical methods in finding relationships in this work [5] where multiple ontologies are used. Though SSP uses a similar idea, it tries to cover documents from various domains. Further, structured data processing and parallel processing [15] are key facts in the field of IR. SSP uses some of these ideas in its data processing.

2.3 Scalability in IR

The **Google search engine** is developed as a large scale hypertext search engine [10]. One of the main goals is to design a search engine that can handle large datasets. Google initially developed a full text and hyperlink database with at least twenty-four million pages. The main idea behind this is the Page Rank algorithm [9]. This work describes succinctly the idea of global ranking of all web pages. They have designed their system by taking into consideration the rate of document addition, and have estimated that the

initial system developed needed to be replicated ten more times to index all documents written by people in the whole of the United States.

In "**Enterprise Search Evaluation Guide**" [18], there is discussion of seven critical characteristics any enterprise '*Search'* solution should have. Scalability is discussed as one of the most important measures. Scalability of an enterprise search solution refers to three key metrics, which are also SSP's primary focus:

- *The number of documents the search engine indexes*. This is the primary metric on which scalability is based.
- *The number of queries per second that the system can serve to concurrent users*. This is another important concern for search engines to consider, so that they can satisfy the end user requirements.
- *The amount of man-hours it takes to administer the solution as it grows*. This is also one of the important factors because there could be many data centers where backups and fail-overs must be properly administered.

Scalability is studied at various levels in the information-processing field. In the clustering domain, Cutting et al. [7] described how to cluster a document collection hierarchy and allow the users to browse through the clustered collection. Also, there is a provision to rescatter the documents. It has been discussed here that clustering should be viewed as an information tool as opposed to a technique to improve conventional search techniques. To support **Scatter/Gather**, fast clustering algorithms are essential. Clustering can be done quickly while working on small groups of documents rather than an entire corpus. For extremely large corpora, the linear time clustering achieved by their algorithms may be too slow. The solution was to develop variations on Scatter/Gather, which will scale to arbitrarily large corpora.

Scalability is important while working with large data sets. IBM's work [12] about scalability using larger systems discusses experiments with a search application using a large data set, i.e., the **Nutch/Lucene application**. The Nutch/Lucene search framework by Moreira et al. have shown that it is feasible to achieve scalability to a great extent in commercial search applications by using an appropriate search tool kit and system set up.

They experimented with nearly 40 gigabytes of documents and have shown good results in terms of fetching relevant documents. They dealt with a larger dataset (key factor for scalability). Execution time is nearly equal to Google's query processing time, which is 2 seconds approximately.

2.4 Scalable Search Frameworks

IBM [13] compares **scaling-up** (using bigger boxes) to **scaling-out** (using more boxes). They use Nutch search as their workload and Nutch uses Lucene as the underlying search tool kit. They concluded that scale-out solutions have an indisputable performance and price/performance advantage over scale-up for search workloads. They also concluded that scale-out systems are still in a significant disadvantage with respect to scale-up when it comes to systems management.

In this work the indexes are replicated and query is run in parallel across all these indexes so that the query performance can be increased. Particularly, the focus was to improve the time taken to fetch the documents from the index after the initial step of locating the references. This is one key aspect, which could be used in scalable SSP to address performance considerations.

"Similarity Search on the Web: Evaluation and Scalability Considerations" by Haveliwala et al. [19] introduced a new technique to perform search. They contributed a novel method to automate the evaluation process. Finally, they also show how to scale this particular work to handle millions of web pages using the established Locality-Sensitive-Hashing technique.

In "Search Engine Technology and Digital Libraries", Nobert Lossau [5] discusses about search engine technologies which can be used in digital library implementations. Since digital libraries deal with a large set of documents there is an emphasis for robust and scalable search technology; 'Fast' technology is used to address this requirement. This is an innovation-oriented method that provides a robust and scalable core technology. The alternatives, which are suggested along with 'Fast', are open source developments like 'Nutch' and the Jakarta 'Lucene'.

Wikipedia, one of the most widely used information sources, provides details about a variety of information on domains, people, and events. It deals with a large amount of data. It supports search, which is based on the Lucene search toolkit. This assures that Lucene is an ideal search tool kit to develop search applications, particularly the ones which need to be deployed in real time. For a list of applications which use Lucene, please refer to the link "http://wiki.apache.org/lucene-java/PoweredBy".

Clustering is the task of grouping related documents in the document collection. The major focus of the research related to clustering documents is to prove that information retrieval effectiveness is tremendously improved by clustering [20]. Clustering deals with the entire document collection. Scalability and other factors associated with this, particularly various measures used to improve scalability features, are discussed in [8, 21, 22].

SSP uses **citations** and **co-citations** of documents to calculate similarity and find relevance of documents in addition to the title and document content similarity. Various studies related to usage of citations [16, 23] assures that the idea of using citations in SSP is appropriate. Fox et al. studied the effect of combining evidence [24], which is important in SSP to support path relations.

Lucene serves as a search tool kit for many web and desktop applications, which are commercially deployed. Wikipedia uses Lucene as its underlying search toolkit. Also some other popular and widely used sites like Jguru, Eclipse, Source Forge, digital libraries, and Ecommerce search sites use Lucene as the basic search toolkit. Wide scale applications of Lucene [25] suggest it as an ideal search tool kit to develop scalable and robust search applications. Lucene can be adopted as the basic search tool kit in SSP. Meij et al. [26] have deployed Lucene in a grid environment, which suggests that Lucene can work quite well for SSP also.

"**Lucene in Action**" by Otis and Erik [11] discusses core Lucene capabilities, Lucene based implementations, and the Nutch search application. Following are some of the reasons for Lucene's success:

1. Lucene offers parallel indexing capabilities and uses many segments for temporarily storing data. The segments are merged to get the final index structure.
2. There is flexibility in choice of the index: RAMIndex, FileSystem Index (FSIndex), or Database Index. This can be chosen as per one's requirement. For a scalable implementation, FSIndex is the appropriate one since there is no limitation on the size of the file system or disk space, which is used for storing the index. SSP can be greatly benefited by huge secondary memory support in current systems.
3. Data can be moved across the RAMIndex and the FSIndex if required by an appropriate set of steps.
4. Parameterising capabilities of Lucene helps to improve the time by using factors like MergeFactor and MergeLimits.
5. Support is available for different type of queries.
6. Parallel searchers and caching queries are also some of the key concepts behind Lucene's architecture, which improves search system capabilities.

Chapter 3. A Collection to Test Scalability of Stepping Stones and Pathways

3.1 Summary

This chapter describes the data set requirement to test the scalability of Stepping Stones and Pathways, based on which the candidate test data set is chosen. Also the steps involved in processing the chosen data set are discussed.

3.2 Scalable SSP Test Data Set Requirements

Following are the data set requirements for Stepping Stones and Pathways. The reason for choosing CiteSeer data is also explained below.

1. Bigger collection:

   The earlier implementation of Stepping Stones and Pathways used a data collection having 6000 documents. Since the goal is to scale this up, there is a need to choose a data set which contains more than 6000 documents. The CiteSeer collection has nearly 700,000 records and satisfies the larger data set requirement to test the scalability of SSP.

2. Collection supporting document relations:

   The key idea in Stepping Stones and Pathways is to find the relationships between concepts in the form of document chains. This requires the collection to have overlapping document information. The CiteSeer data set serves as an appropriate source repository since it includes documents and relationships from related fields in the Computer Science domain.

In the field of Computer Science there is ongoing research on databases, information retrieval, operating systems, networks, algorithms, and programming languages. The CiteSeer documents contain information about the above-mentioned Computer Science topics. So this test collection could serve as a good test data set to discover various interesting associations among topics in terms of document relations.

3. Documents with references or citations:

Stepping Stones and Pathways uses citations for analyzing the relationships among documents and forms a part of document similarity calculation. This requires the data set to have the citation information. The CiteSeer data set contains information about the references (citations and co-citations). Since there is a close match between the scalable SSP data set requirement and CiteSeer's nature and size of the data set, it can be used as the test data set.

3.3 Data Set: CiteSeer Collection

The scalable SSP implementation is tested using the CiteSeer data collection. CiteSeer is a scientific literature digital library, which supports autonomous citation indexing, awareness and tracking, citation context, and related documents. CiteSeer's goal is to improve the access of academic and scientific literature. The information about papers is available as XML metadata, which is much simpler to process. Since the nature and size requirement for a scalable SSP test data collection is satisfied by the CiteSeer data set, the same is used in testing the new scalable SSP system.

3.4 Steps to Process the CiteSeer Metadata

The CiteSeer data is processed using the following steps to extract the data and create an index.

- The CiteSeer site is a repository of scientific papers related to research in computing. CiteSeer has the XML metadata corresponding to each of these documents. The metadata information include the high level information about the paper like title, description, author, date, identifier, source, references, subject, etc. These metadata dump files can be downloaded from the CiteSeer site.
- Every dump file contains nearly 10,000 records, each of which is in the format shown in the following section. To process the XML metadata documents, the data is extracted using an XML parser and information is indexed.

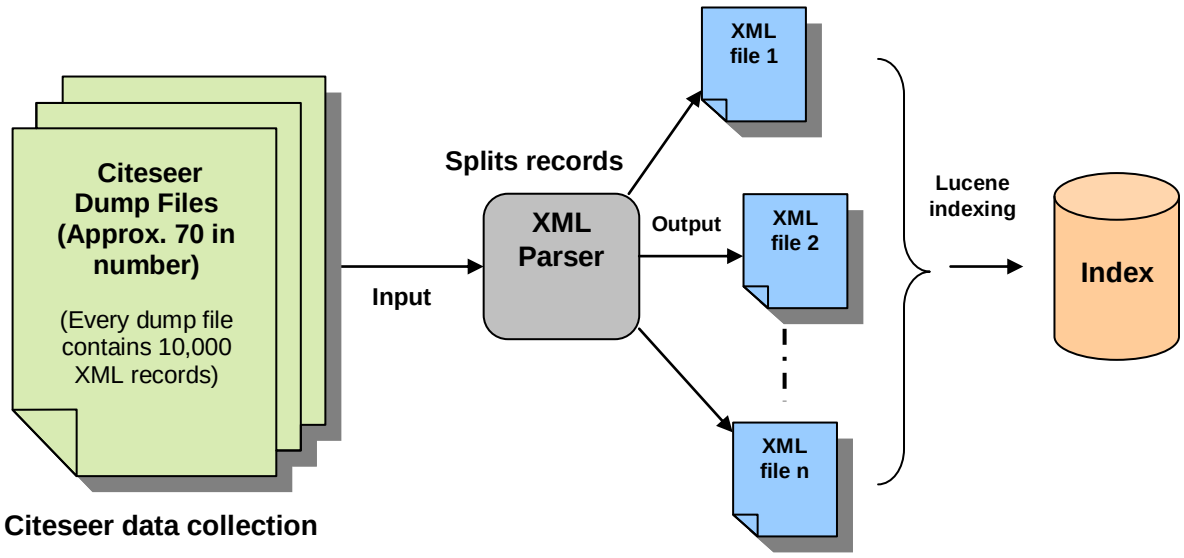

Figure 5: CiteSeer Data Processing

3.5 Sample XML Input to Test the Scalable SSP Implementation

Following is the sample metadata of a single XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <record>
    <header>
      <identifier>oai:CiteSeerPSU:620025</identifier>
      <datestamp>2003-06-11</datestamp>
    </header>
    <metadata>
    <oai_CiteSeer:oai_CiteSeer
xmlns:oai_CiteSeer="http://copper.ist.psu.edu/oai/oai_CiteSeer/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

xsi:schemaLocation="http://copper.ist.psu.edu/oai/oai_CiteSeer/
http://copper.ist.psu.edu/oai/oai_CiteSeer.xsd ">

<dc:title>Robust Induction of Process Models from Time-Series Data</dc:title>

<oai_CiteSeer:author name="Pat Langley"></oai_CiteSeer:author>

<oai_CiteSeer:author name="Kazumi Saito"></oai_CiteSeer:author>

<dc:subject>Pat Langley,Dileep George,Stephen Bay,Kazumi Saito Robust Induction of Process Models from Time-Series Data</dc:subject>

<dc:description>In this paper, we revisit the problem of inducing a process model from time-series data.</dc:description>

<dc:date>2003-06-11</dc:date>

<oai_CiteSeer:pubyear>unknown</oai_CiteSeer:pubyear>

<dc:format>ps</dc:format>

<dc:identifier>http://CiteSeer.ist.psu.edu/620025.html</dc:identifier>

<dc:source>http://www.isle.org/~langley/papers/process.icml03.ps</dc:source>

<dc:language>en</dc:language>

</oai_CiteSeer:relation><oai_CiteSeer:relation type="References">

<oai_CiteSeer:uri>oai:CiteSeerPSU:27335</oai_CiteSeer:uri>

</oai_CiteSeer:relation>

<oai_CiteSeer:relation type="References">

<oai_CiteSeer:uri>oai:CiteSeerPSU:451031</oai_CiteSeer:uri>

</oai_CiteSeer:relation>

</oai_CiteSeer:oai_CiteSeer>

</metadata>

</record>

</root>

The XML data provided above contains details about the technical papers in the field of Computer Science. This metadata contains high level details in technical papers like title, author, date, abstract, and references. In the above input, only the data represented in bold face elements are processed. This could serve as potential search data for user queries.

Following are the XML data elements that are extracted, tokenized, and indexed:

1. Identifier:

   The identifier is the unique ID of each paper. It has the prefix *"**oai:CiteSeerPSU:"*** followed by a unique numerical identifier value.

   <identifier>oai:CiteSeerPSU:620025</identifier>

2. Title:

   The title provides information about the paper topic.

   <dc:title>Robust Induction of Process Models from Time-Series Data</dc:title>

3. Description:

   The description gives brief information about the paper.

   <dc:description>In this paper, there is a need to revisit the problem of inducing process model from time-series data.</dc:description>

4. References:

   The references are of two types

   i. Citations, which this paper cites or references to.
   ii. Co-Citations, which are the set of papers, which refers this particular paper.

<oai_CiteSeer:relation type="References">
<oai_CiteSeer:uri>oai:CiteSeerPSU:451031</oai_CiteSeer:uri>
</oai_CiteSeer:relation>

When calculating the relevance between documents in the initial SSP system, the reference information is also used as a part of the similarity calculation as it increases the effectiveness in search.

3.6 Candidate Collections

The information retrieval field includes diverse methods and systems that implement various IR concepts. Since each project focuses on a specific area of IR, the test collection requirements also vary. The nature and amount of data in each collection would vary depending on the measurable factor. In the case of scalability projects, larger

data collections need to be used. Apart from CiteSeer, other data sets, which can be used for testing are discussed below:

Blog Data Collection:

The blog data collection has information such as blog URL, name of the blog, description about the blog, links in the blog, etc. However, this collection does not contain citations and co-citations. It is difficult to form good clusters unless there are many blogs discussing the same topics. Though this data set may appear to have similar data structure to CiteSeer it does not form a good candidate set for testing.

Wikipedia:

Wikipedia is a huge information source, which has been used as a data set in the Omnipelagos, which is one of the commercially deployed connection finding search engines. The information available here is mostly about entities like people, places, and events. This data set could help in discovering transitive relations to a great extent due to the nature of document information overlap.

Chapter 4. Architecture

4.1 Introduction

This chapter discusses the overall architecture of the scalable Stepping Stones and Pathways system. The new architecture is very similar to the initial SSP system, except for the major change in the SSP engine. The back end (search engine part) is replaced with a new scalable one. The new search engine has a number of tools and jars.

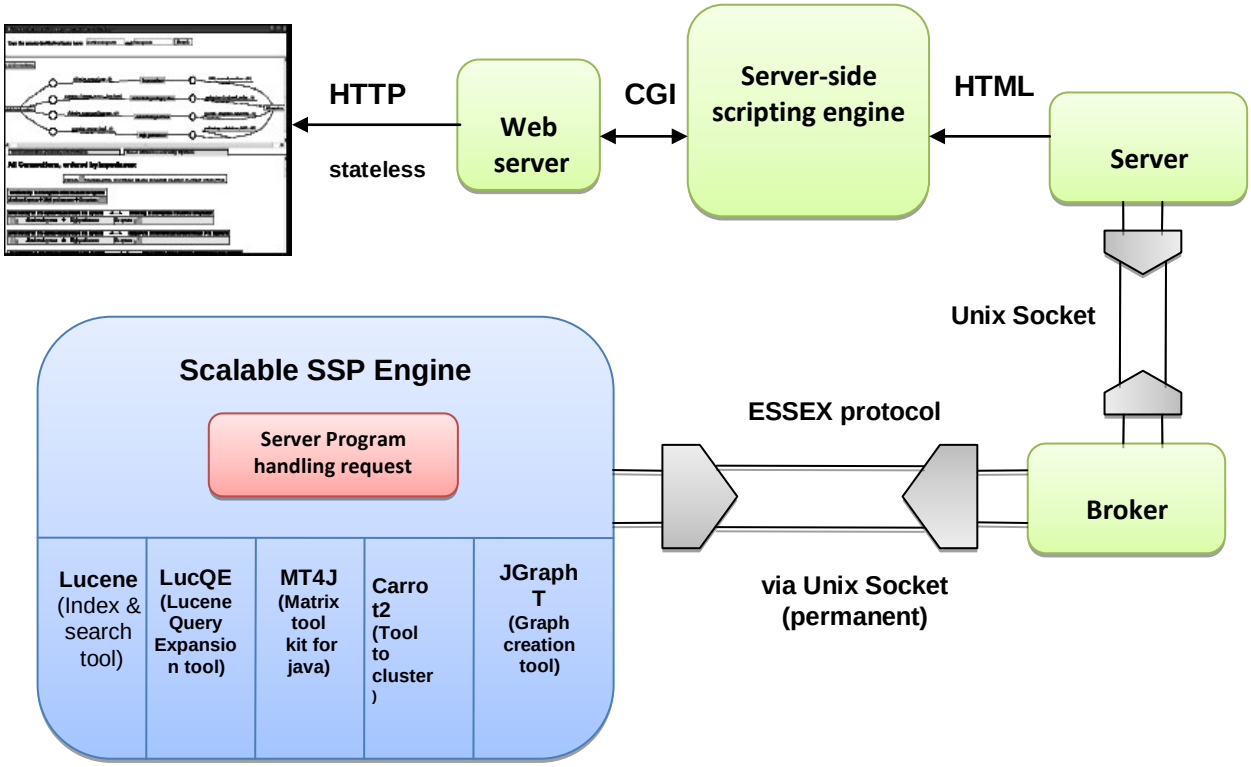

Figure 6: Architecture of Scalable SSP

4.2 Components of SSP

The architecture of the scalable SSP is shown in Figure 6. The major components of the SSP framework are

1. Back end (i.e., search engine)
    a. Offline processing
    b. Runtime processing
2. Broker
3. Client

4. Web components
    a. Server
    b. User interface

The architecture of scalable Stepping Stones and Pathways is similar to the initial model except for the back end part where there is a major change. In the back end (search engine) there are many new components, which have been added or changed. The SSP architecture is divided into the following parts:

- The User interface is the front end part, and it runs inside a web browser.
- A Web Server accepts the requests from the user interface and passes them to the programs that process these requests. This is done by the server pages like Java Server Pages.
- A Server-Side scripting engine takes a server page, executes the code, and sends the result back to the web server. This is done by the server side programs like Servlets.
- The Server Page uses the command which comes as a part of the URL string and contacts the search engine for further processing through the Broker. It presents the results which are given by the search engine to the user interface.
- The Broker maps session identifiers in the URL to a user session in the search engine. It passes the commands to the search engine in an appropriate format. Also it reformats the results returned by the search engine.
- The Search Engine implementing SSP as a retrieval method retains the state of a user session. In the new architecture, the old components have been replaced with new scalable components to address the scalability issue in the initial system.

4.2.1 Back End

The back end is the key component where the query processing occurs. The initial search engine had a limitation to handle only 6000 documents. An in-memory index was used, which involved constraints to keep the index small. The new architecture addresses this issue by providing scalable components in the back end. This includes the Lucene scalable index and search toolkit, the matrix creation tool mt4j (matrix tool kit for Java), the query expansion tool LucQE, the clustering system in the Carrot2 framework, and the graph creation tool JgraphT.

The back end takes two input queries, processes them, and presents the result to the client through the broker. It includes the offline processing tasks like indexing and matrix creation. The runtime query processing also happens in the back end.

4.2.2 Broker

When the user starts the Stepping Stones and Pathways system, session identifiers are created. The Broker is a Unix daemon that performs the job of mapping session identifiers to Essex sessions. It waits for user requests in a particular Unix socket and every user request must include a session identifier string. If the session identifier is not present in the pool of session identifiers handled by the broker, then the broker opens a new connection to Essex via a Unix Socket. It keeps the mapping from this session identifier string to the relevant Unix socket. The Broker is also responsible for timing out user sessions.

4.2.3 User Interface

The user interface in SSP consists of both HTML and JavaScript. The web-based user interface offers SSP to the widest possible audience (as long as the user has a modern web browser, they can use SSP). Hence, it is implemented as a web-based application. SSP follows the web-based approach, to reap the benefits of the simplicity of HTML in prototyping, and to allow remote testing without the user having to install software. Also there is no constraint to run SSP under a particular operating system.

4.2.4 Server

At the server side, each web page contains both HTML and the scripts to be executed by Spyce. Spyce is a server-side scripting engine which executes Python scripts embedded in the web page. Each user action in the interface state requires interaction with the back end and it is mapped to web page requests. The script inside each web page performs the following tasks:

- Connects to the broker's Unix socket
- Sends a request through it for an operation to be performed by Essex (for example: create a stepping stone)
- Reads the answer
- Closes the connection to the broker
- Dynamically formats the answer to display it as an HTML page

4.2.5 Graphviz

The scripts inside web pages map the graph description returned by Essex to a visual representation of the connection graph and users can interact with this connection graph. The graphical representation of the connection graph is created by using Graphviz, a graph layout and rendering program. The graph is then post-processed to map available user actions on graph nodes to URLs.

4.3 Steps in Scalable SSP

The steps that occur in the back end part of the scalable SSP implementation are shown in Figure 7. Chapter 5 discusses the implementation steps in detail.

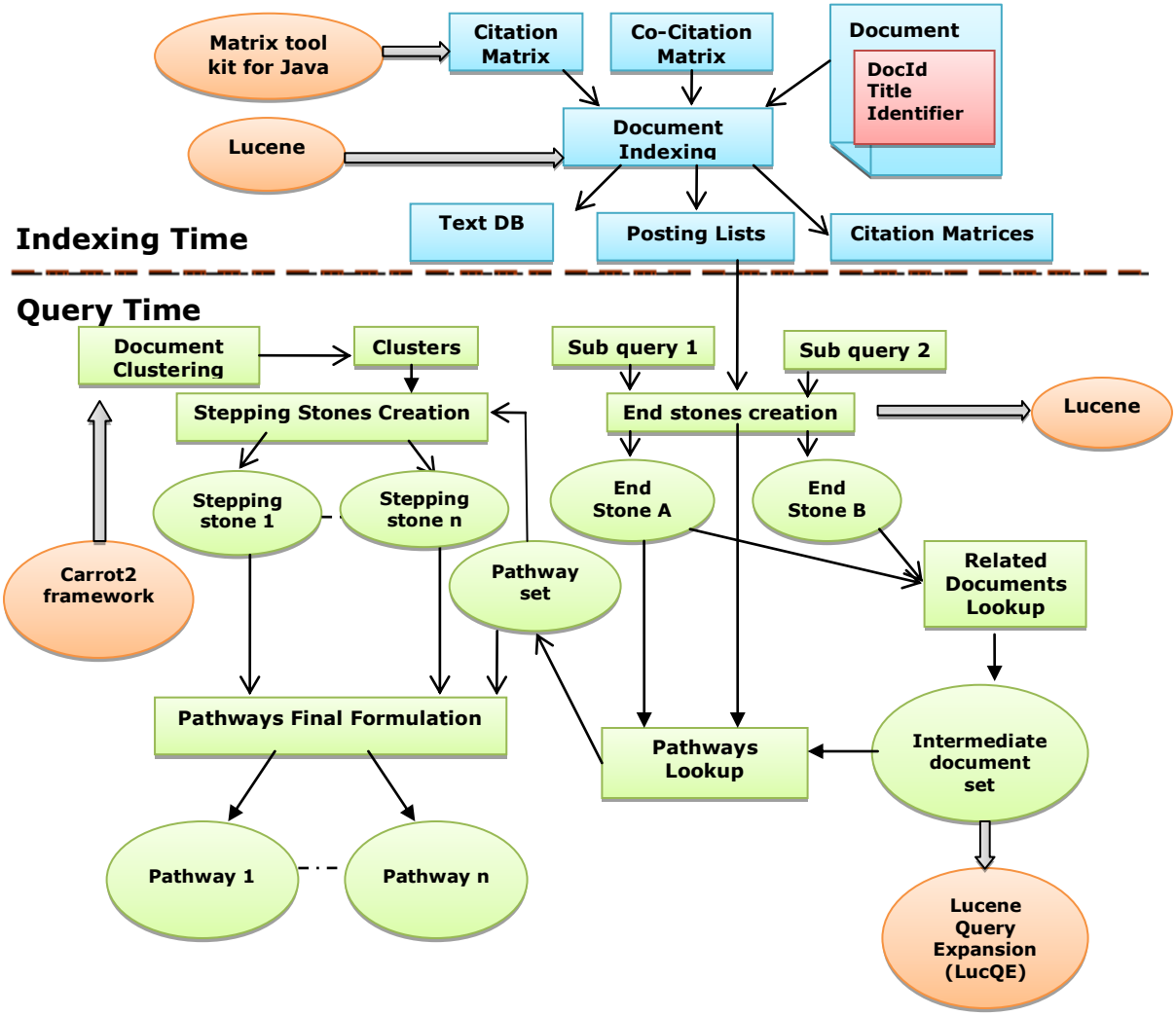

Figure 7: Steps in Scalable SSP

Chapter 5. Implementation of Scalable Stepping Stones and Pathways

5.1 Introduction

In this chapter, the implementation of the scalable Stepping Stones and Pathways is covered in detail. First, the algorithm is reviewed and then each step is explained in detail.

5.2 Scalable SSP Algorithm

The steps to create a scalable SSP are as follows:

1. Download the collection documents and create the index.
2. Use the references in the CiteSeer records, build citation and co-citation matrices.
3. Process query, trying to match all words in the sub-queries first, and if that fails, relaxing the sub-queries by making words optional.
4. Create end stones:
    a. Retrieve documents for each input query.
    b. Create two clusters (called centroids), one for each document set.
    c. Calculate the cluster centroid from the top documents in the cluster.
5. Create Stepping Stones and Pathways:
    a. Use the end stone centroids and form the expanded query. This is executed to get two document sets, one for each expanded query.
    b. Create an intermediate document set with the documents that appear in both retrieved sets.
    c. Find relevant connections between the documents at the endpoints and the documents in the intermediate set.
    d. Eliminate all documents in the intermediate set that are not part of a connection.
    e. Cluster and label all documents left in the intermediate set. These clusters become topics.
    f. Present all pathways (documents + topics to which they belong) to the user.
6. Explore documents, topics, and pathways.
7. Add, remove, join, and split topics.

5.3 Preprocessing/Offline Task

The information retrieval field works with large sets of documents. *Search* mainly focuses on indexing and querying tasks. The main idea in *search* is to keep the query processing time to a minimum. To achieve this, the tasks are analyzed and classified as either offline task or runtime task. If the number of documents, which we handle, is on the order of millions/billions, for effective search, the following tasks can be performed offline:

1. Index creation.
2. Matrix creation using the references or links.
3. Clustering the documents.
4. Training the query/collections.

In SSP some of these tasks are done offline.

5.4 Document Indexing

Document indexing is an offline task wherein an index is created from a set of XML documents. In this task we process CiteSeer dump files, which are a collection of XML records about scientific papers in the field of Computer Science. There are nearly 70 CiteSeer dump files each containing 10,000 XML records. Each record has elements containing information about a paper like title, description, references, author, etc. The task is to extract tags, which might contain potential search terms, tokenize, analyze using appropriate analyzers, and index the terms.

The main focus in the scalable SSP implementation is to handle huge index creation. This requires large storage space. In order to deal with such a large index the file system is chosen as opposed to main memory, which has memory limitations.

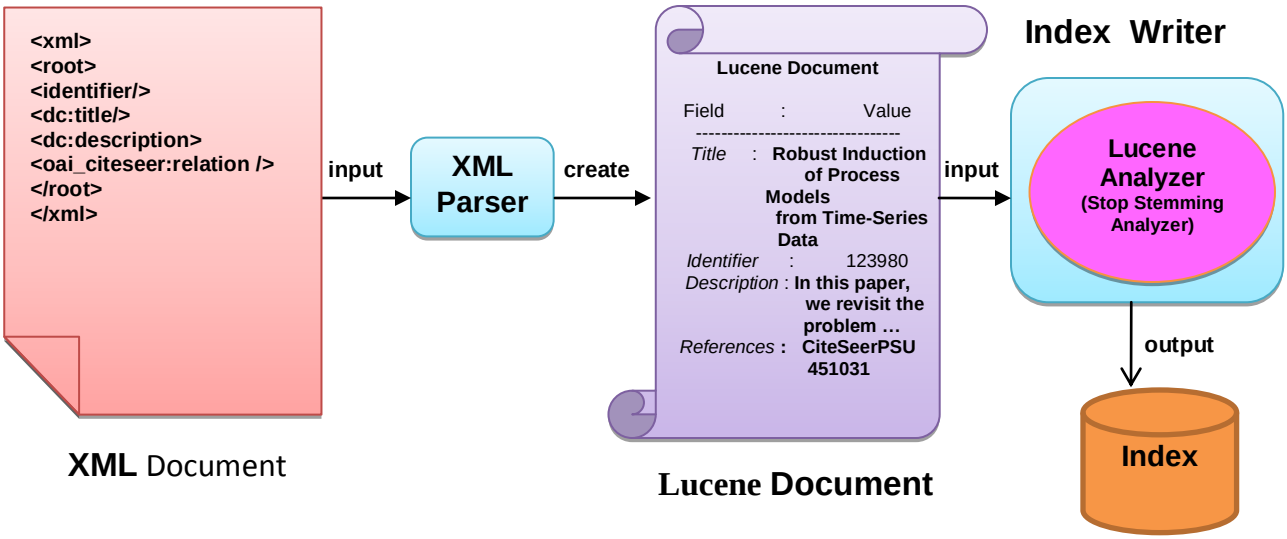

Figure 8: Document Indexing

The above process involves the following steps.

5.4.1 Splitting XML Records

The CiteSeer collection has huge metadata files. One reason for splitting these files into smaller ones and then processing the XML data elements is to make the processing simple. Another reason for splitting is, some records have data issues (mentioned in the issues section) because of which the huge dump files cannot be parsed at one go by DOM Parser. Thus we split each dump file having 10,000 records into smaller files each having just one record element. The format of a single record is explained in the test collections section. With this processing, the records which are not well formed or have character-set issues, can be detected. Mark these records by logging the information about the files, which cannot go through parsing, and proceed with processing other records.

This process is done for each of the 70 dump files. As an output of this splitting task, create the following:

1. Directories with the name of dump files
    a. For example: oai_CiteSeer10.dump/.
2. Inside each such directory, split and place all the records from a dump file that has the same name as the directory. The original dump file contains

around 10,000 records and they are split into smaller files, each with the name format like: rec1.xml […] rec10000.xml.

5.4.2 Indexing: Lucene Document Creation

Indexing is the task of parsing the document and creating an index with the tokens. The tokens form the key along with the posting lists containing other information like number of occurrences of that word and proximity information. First, process every recXYZ.xml and extract the data from the elements, which needs to be indexed. Secondly with the extracted data, create a Lucene "Document" structure, which have the field and its corresponding value. We assign the element names as field names and extracted contents of the element as values. The following tags are indexed from the XML records,

- Title
- Identifier
- Description
- References

5.4.3 Processing Document using IndexWriter

After creation of the Lucene document we pass the document to the IndexWriter, which processes and creates tokens and other relevant data as a part of the index structure. The addDocument() method of the IndexWriter is the main API which is used to achieve this. Lucene performs the index creation by creating small segments initially and merging them finally into a bulk index segment. This bulk index file is stored in the file system directory chosen to store the index. After all records are indexed, the IndexWriter object has to be optimized and closed. Since the index is a shared object, synchronization principles should be followed while reading/writing the index. In addition, it creates a TermVector as a part of this index, which makes document comparisons much simpler.

5.4.4 Verifying the Index

The index created above can be verified for the structure, tokens, most frequently occurring words, number of records, and flag parameters. Flag parameters indicate what is done with the tokens once they are parsed. Luke is a graphical user interface (GUI) tool that is used to verify the index, which is created. The following diagram shows the

overall index structure with the tokens, number of occurrences of tokens, and top "n" number of tokens along with the option to select fields.

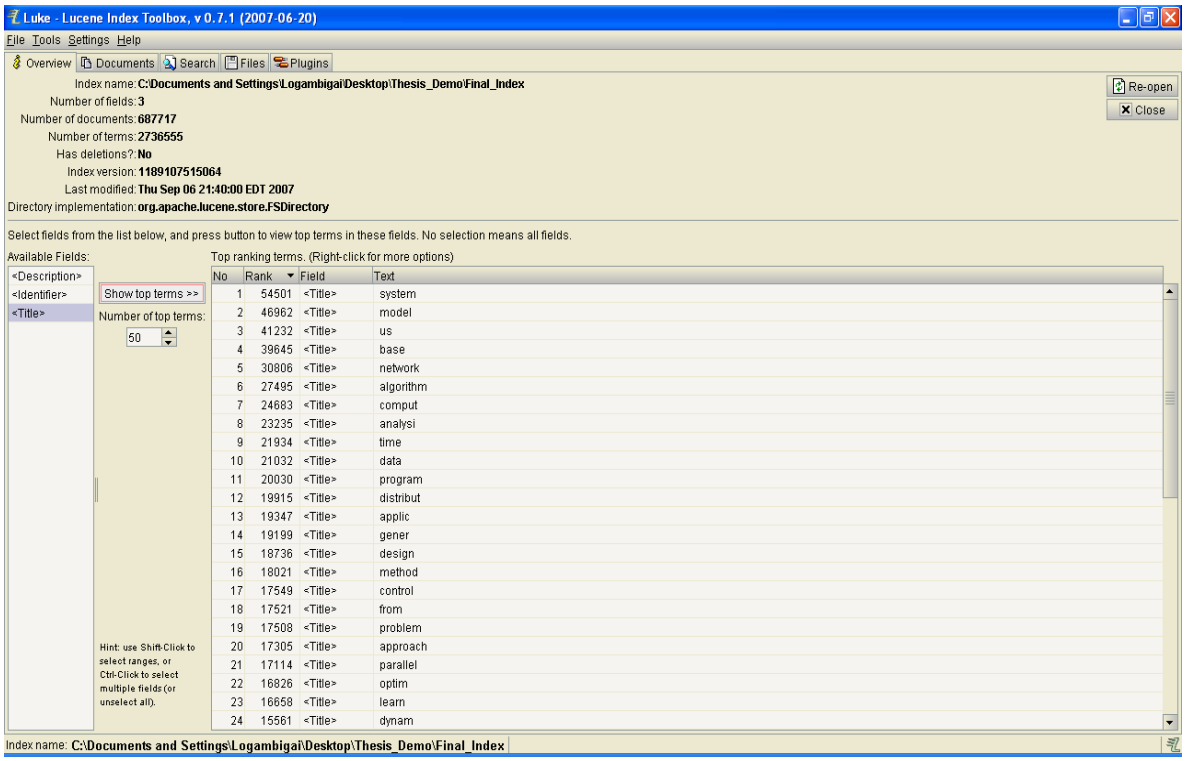

Figure 9: Lucene Index Overview

Along with the above options there are other features like document browsing where the indexed document can be browsed. Lucene allocates an internal document number for every document which starts from 0 and ranges to the number of documents minus one. Also, as part of the document detail these flags are included: Indexed, Tokenized, Saved, TermVector etc. Figure 10 shows the detailed view of the documents.

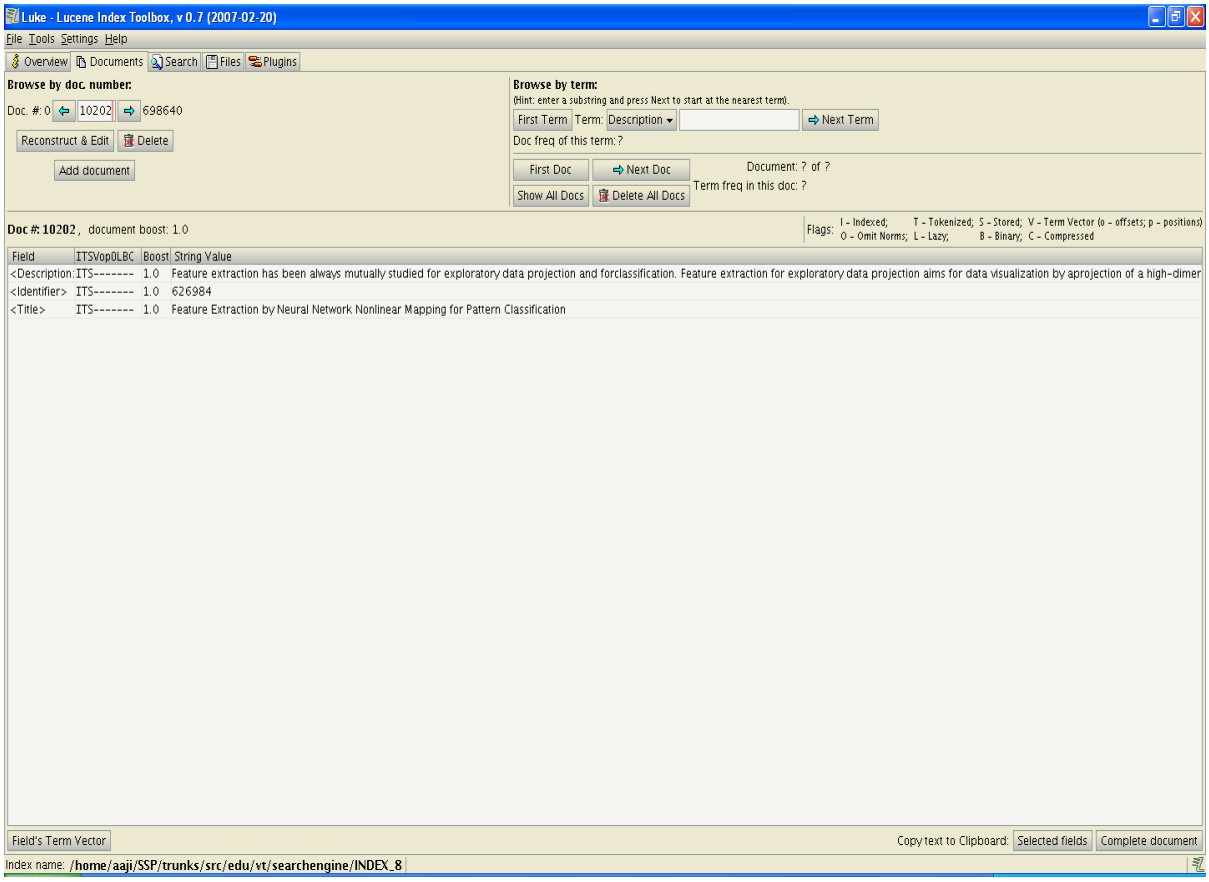

Figure 10: Documents Indexed using Lucene

Luke provides search and analyzers. In the search tab, the phrases to search can be entered and one can choose the field on which this search has to be performed. There are many analyzers, which are available as a part of Luke. Different analyzers can be chosen to understand how search results differ with the choice of the analyzers.

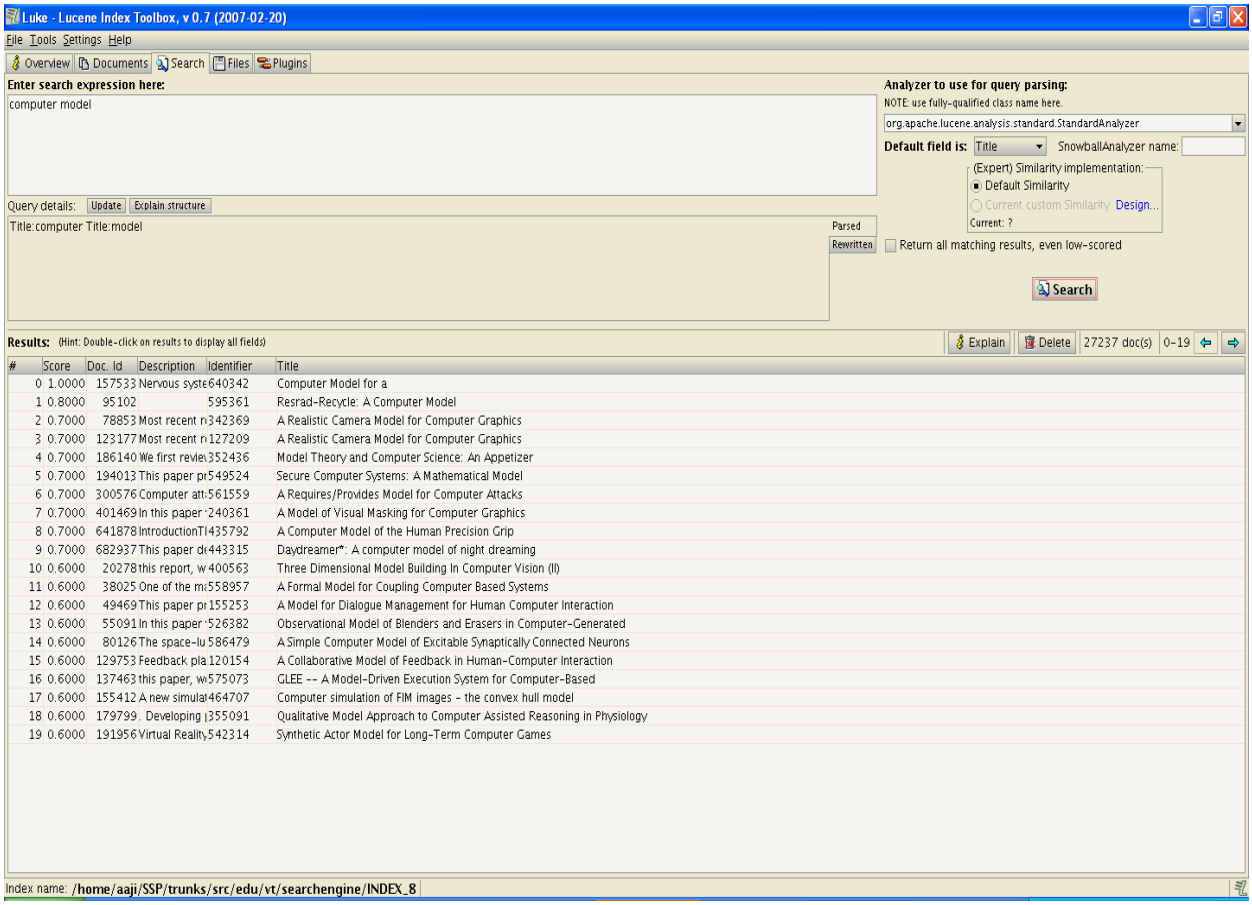

Figure 11: Search using Luke (Lucene based)

5.4.5 Tokenizing using Analyzers

The analyzers are used to process a string of text and create tokens. The Lucene analyzer includes many tokenizers and filters. The tokenizer breaks the text into words and whitespace. The filters do the post-tokenizing, which is removing stop words, stemming, changing case, and so on. In our case the StopStemming analyzer is used as part of indexing. This uses a list of stop words that need to be removed, since the common English words might not be useful for searching.

5.4.6 Incremental Indexing

The CiteSeer collection grows with the addition of new research documents. The index which is created initially is extended with the new documents. Lucene supports incremental indexing where the new documents are appended to an already existing index. This helps in minimizing the time taken to update the index with new records.

5.5 Citation/Co-Citation Matrix Creation

In this module, we follow a similar approach as indexing wherein we parse the dump file and extract the contents of the identifier and reference tags from each XML record. This can be used to build the citation and co-citation matrices, which will essentially be sparse matrices. From the sparse matrices, the set of document IDs can be fetched that are citing a particular document and vice versa.

5.5.1 Matrix Creation Steps

The matrix creation for the citation and co-citations involves the following steps.

1. First find the maximum value of the identifier that is available among all the dump files. This is required to set the size limit for the sparse matrices.
2. In all the dump files, grep for all the <identifier> and <oai_CiteSeer: uri> tags which contain identifier, and redirect the output to a temporary file.
3. After collecting all the identifier values in a file, find the maximum identifier value and store that value in <SSP_Path>/trunks/conf/citationMatrixSize.txt
4. Create sparse matrices of the above maximum size, and store the citation and co-citation matrices in the following file:
   <SSP_Path>/trunks/src/edu/vt/searchengine/Output/ CitationMatrix.txt

5.5.2 Citation and Co-citation Matrices Overview

The Citation and Co-Citation matrices are used to store the references; these are used in the document similarity calculation. The citation matrix for the collection is created as a one-dimensional array of identifiers. In Figure 12 this array is marked with the label row numbers, which start from 0 and end with the value of the maximum identifier. When a paper cites a set of papers, those are shown in the linked second dimension, which is marked by column numbers. In this example, shown in Figure 12, the document with identifier 2 and 115 have a set of references, which are shown by the column numbers. The space optimization is achieved by following such a storage layout, i.e., the sparse matrix structure.

A paper might have been referenced in many other papers. The papers which have been referenced are the Co-Citations. The Co-Citation matrix shows the identifiers of papers in

the first dimension, which are being cited in other papers. In Figure 12 the array, which has been referred to as the primary array, describes the set of papers in which a particular paper is cited.

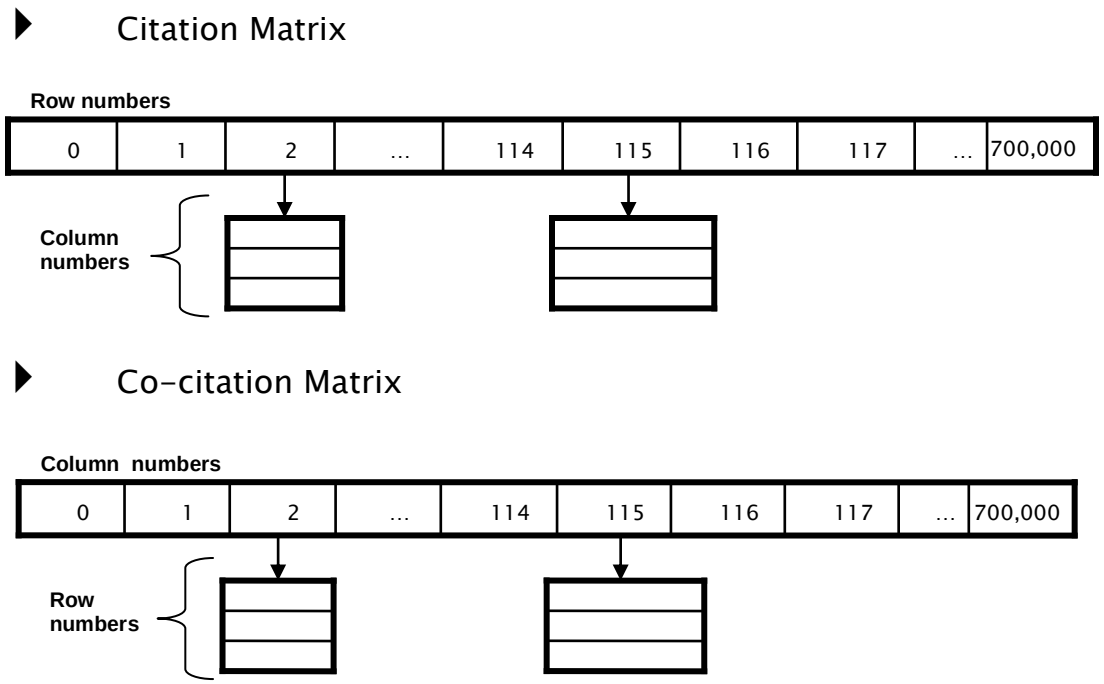

Figure 12: Sparse Matrix in Lucene

5.6 Query Time Tasks

5.6.1 End Stones Creation

In the end stones creation, we query for the relevant document set for each sub-query A and B. Lucene offers an option to limit the number of documents which are retrieved as a result and we have set it to 12 for our implementation. The documents, which are retrieved in Lucene, are called the TopDocs. Once the documents are retrieved, calculate the centroid for each TopDocs document set. The centroid is the collection of the top 35 terms, which occur most frequently in the set of documents returned as a result of the search. These words form the expanded query, which get the potential set of connection documents. The search using the centroid query gives a set of documents which forms the intermediate document set.

The citation vector and a co-citation vector for the newly retrieved document set are calculated. At the end of this step, the common set of documents is fetched for both subquery A and B, which will be used further by the next set of modules: Related Document Lookup and Pathways Lookup. The steps detailed below are followed for the end stone creation.

1. **Loading Matrix**
    - The matrix, which was created in the above module, is written to a file called Output/CitationMatrix.txt. Using MatrixVectorReader we read this file. Further we get the i) Matrix information and ii) The matrix size from matrix info. This is a one-time process since this matrix is reused in later steps.
    - Initialize the FlexCompRowMatrix with the above matrix size.
    - Loop through for MatrixSize times and read the line from the reader. Create a sparse vector reading the array of data from the reader and set it as FlexCompRowMatrix Row.
    - In this manner populate both the row and column.

2. **End Stone Vertex Creation**
    a) Create an IndexSearcher on a particular directory.
    b) Call the search, which will return the TopDocs.
    c) Create the Citation and Co-Citation Vectors from these documents (will be used by later modules).
    d) Calculate the query centroid, which returns the expanded query.
    e) Perform a search using the centroid query and get the expandedTopDocs.
    f) Calculate the ExpandedCitation vectors
    g) Get the union of documents from the
        - TopDocs
        - Expanded Docs
        - Citation Vectors

- Co-Citation Vectors

5.6.2 Query Expansion

The TopDocs, which are retrieved by running the sub queries, is used as the input to get the expanded query. The Lucene query expansion library is used to generate the expanded query. The QueryExpansion is the main class, which is instantiated using the following as input: i) Analyzer, ii) IndexSearcher, iii) DefaultSimilarity, iv) Properties. The expandQuery() method of the QueryExpansion processes an original query string, and documents returned for the initial query. It gives the expanded query as output. The expanded query contains the top 35 frequently occurring words in the set of TopDocs.

5.6.3 Related Documents Lookup

The output of the end stones creation module serves as the input to this module. Here, search for relevant document matches given each centroid c and the citation and co-citation vectors, using the following formulae:

$$sim(c,d)=1-(1-Pw(c,d)) * (1-Pcocit(c,d) * (1-Pcit(c,d))$$

Take the common set as a potential intermediate set S'. This, along with the previous module's outputs, provides the input to the Pathways Lookup module. The following set of steps is performed to get the set of intermediate documents.

1) The identifier and the score value for the documents belonging to the set 2(g) are stored.
    - The citation similarity score is calculated by using the cosine similarity of the citation vectors of each document.
    - The co-citation similarity is calculated by using the following formula:

$$\text{Score} = \frac{\text{Set size of (A Intersection B)}}{\text{Set size of (A Union B)}}$$

where,

- A is the number of co-citations of the document *c*.
- B is the number of co-citations of the document *d*.

- To find the word similarity between two documents, we use the boolean match of the document titles, and cosine similarity of the document bodies.
- Document content similarity score is obtained from the TopDocs data structure.

The maximum of the above two scores is finally stored as the score of the document.

2) The common documents are put together in one final common doc set (say S') by taking common identifiers from data structures, which are created in the above step.

5.6.4 Pathways Creation

The pathways creation is the step after the end stones and intermediate documents creation. These two sets of documents are used to formulate the pathways.

The expanded query is run to retrieve a fixed set of intermediate documents. These are the potential set of documents, which would connect the two input user concepts. Some of the important steps, which we perform with the intermediate document sets, are:

- Perform the document-document similarity calculation to rank document pairs, which are used in the pathways construction.
- Document length normalization.
- Fixing a threshold for the intermediate documents similarity value.

Before calculating the actual pathways, the intermediate document set is taken and document-document comparison is performed to get the similarity information of the possible, unique document pairs, which are there in the intermediate set.

Document Comparison

The document-document similarity calculation is performed by using the vector space model's cosine similarity approach. Before the comparison, the document length normalization is performed. The index that is created to store the terms contains the following information, which helps in performing the document-document similarity calculation:

- Terms
- Term frequencies in term vectors
- Document frequencies of terms

Document Length Normalization

To handle the anomalies caused by the different document lengths, perform the normalization before computing cosine similarity. This step takes the IndexReader and the term frequency vector and does the appropriate normalization; and returns the value, which is obtained as a result of the normalization.

5.6.5 Intermediate Document Set Comparisons

The comparison of all documents in the intermediate set is done to find similarities for all possible distinct combinations of documents. This results in many document pairs being repeated for this calculation. For example (a, b) and (b, a) is a repetitive comparison for documents a and b. Since it is not required to repeat such comparisons there are some appropriate conditions to avoid the duplicate pairs. Following are the steps that are involved in scoring the intermediate document sets:

1. Compare all distinct document pairs, calculate their similarity weights, and store them.
2. Perform the weight calculation. For this get the following information by performing the following:
    - Get the index
    - Get Term Vectors
    - Get Terms

- Get Term Frequencies

3. Do a Cartesian product of Term Vectors of every pair that is compared and get the weight.
4. Store the information in DocSimilarityData object (a data structure to hold similarity information).

After having the end stone documents and the intermediate document sets, perform the necessary comparisons to formulate the pathways. There are 4 cases, which are discussed with regard to pathway formulation.

Following are the different pathways, which are created, in Stepping Stones and Pathways:

Case 1:

In this case, all the documents are the same across the 3 sets (2 end stones and intermediate document set). This document contains all the terms from the query and they all occur most frequently in this document. Hence this is the highly relevant document for the two input queries. The letter "T" represents the end stone documents and "d" represents the intermediate document set. In case 1 both T and d are the same.

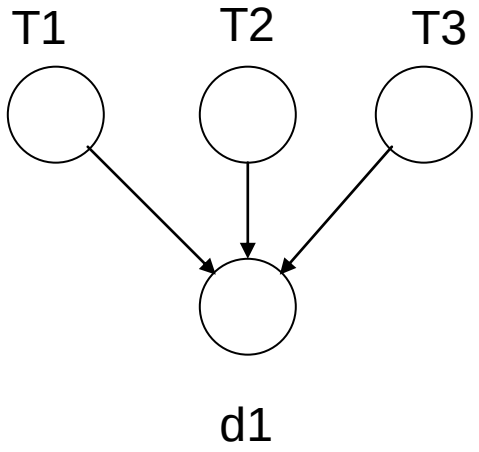

Figure 13: Pathways Creation Case 1

Case 2:

In case 2 the first document occurs in one of the end stones and the intermediate document set. The other document occurs in another end stone and also in the intermediate document set.

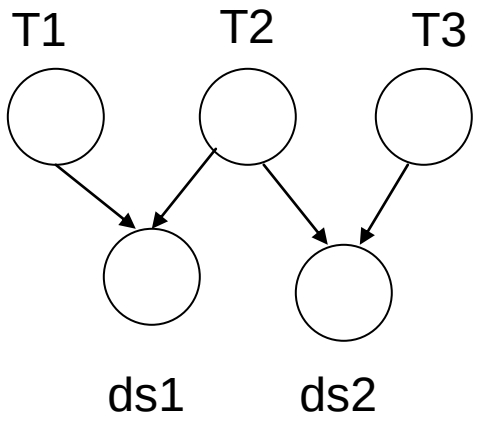

Figure 14: Pathways Creation Case 2

Case 3:

In case 3 one document (T) occurs in end stone set one and in the intermediate document set. Between the other two documents, one of them belongs in the intermediate document set and the other one occurs in the end stone set 2. These two documents have high similarity score compared to all other pairs of documents which occurs in both these sets.

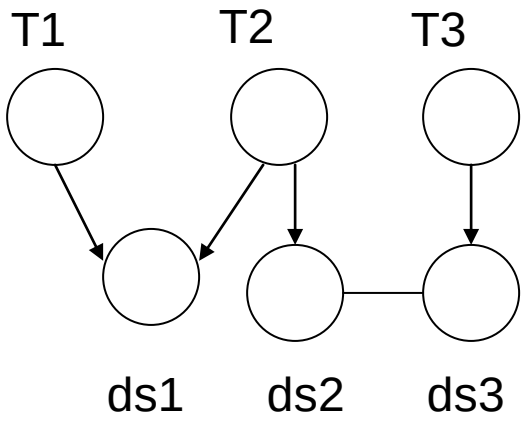

Figure 15: Pathways Creation Case 3

Case 4:

In case 4 one document (T) occurs in end stone set two and in the intermediate document set. Between the other two documents, one of them belongs to the intermediate document set and the other one occurs in the end stone set 1. These two documents have high similarity score compared to all other pairs of documents which occurs in both these sets.

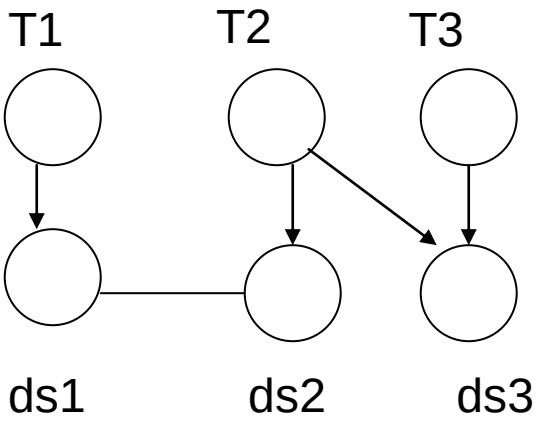

Figure 16: Pathways Creation Case 4

5.6.6 Clustering

Clustering is the process of grouping documents based on the commonness of the subject they cover. The intermediate document set is clustered to group the documents according to the field of topic. The current clustering mechanism have limitations in handling the entire collection set at once. So currently runtime clustering is used where the clustering algorithm is run on the intermediate documents and they are grouped. During clustering, labels are assigned to the documents, and this is the way Stepping Stones are created. These labels are assigned to the pathways when the final pathways are formulated.

5.6.7 Pathways

The top paths are the ones which have high relevance to the query and they are formulated using this step. The pathways created contain the document identifier for all the documents as a part of the pathways and score of the pathways, which is the total of all the similarity scores of the pathways documents. To get the final pathways in the UI we select the top scored pathways from the collection of the calculated pathway set.

5.6.8 Assigning Cluster Labels

The clusters which are created contain a set of documents grouped by a common topic and labels assigned to each cluster. The pathways, which are to be displayed, are tagged with cluster labels. This involves taking the pathways document identifiers and checking the clusters to which they belong and assigning appropriate cluster labels.

5.7 Integration with User Interface

The integration part connects the search engine and the UI. This is responsible for sending the results from the back end to the front end for display. The front end has two parts. First one is the graphical part which displays the chain of documents as graph and this appears in the upper part of the UI page and second one is the detailed table of pathways, which appear in the lower part of the UI. The information, which is displayed in this graph, is:

- End stone and stepping stone topics/query as the nodes.
- Links displaying the connections between the nodes, and labels which are the terms that are common between the nodes that are linked.

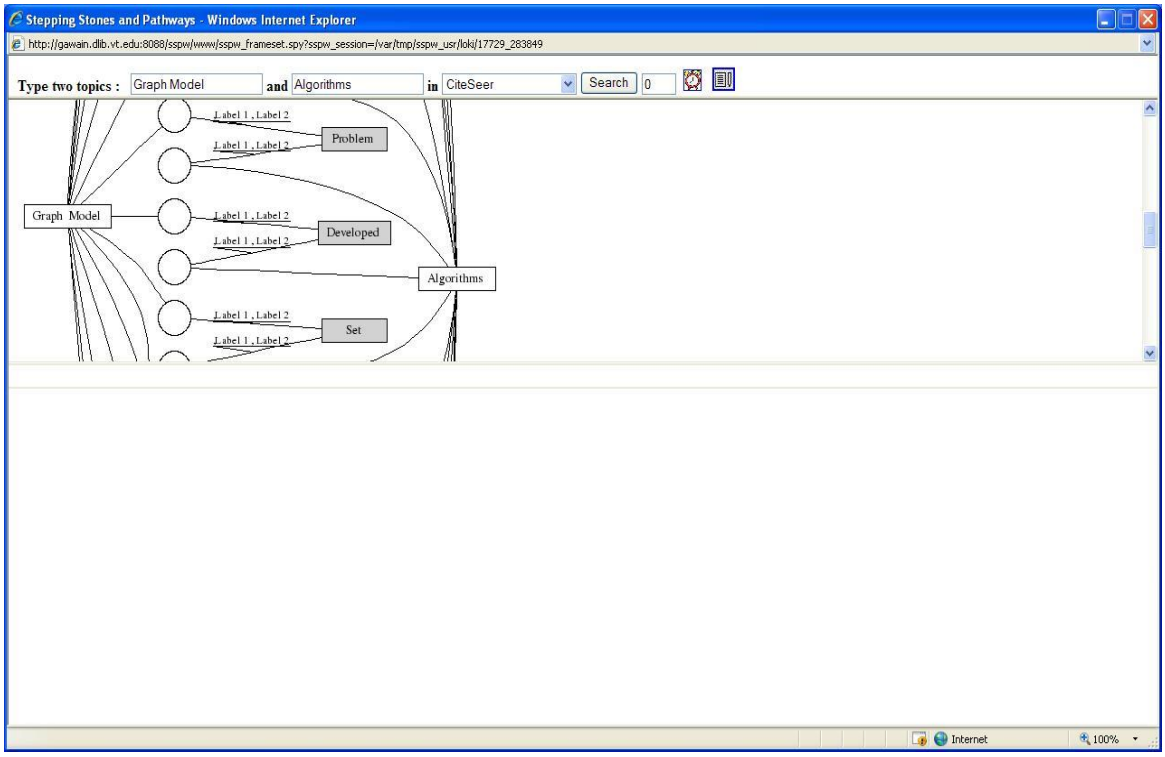

Figure 17: Stepping Stones and Pathways Graph

The lower part of the UI page contains the documents corresponding to the graph. This part is divided into two parts. The first part contains the pathways information, which has the following data:

- Title of the documents, which are connected
- Document identifiers

For the above pathways there may be many cluster associations. They are displayed in the lower part of the graph.

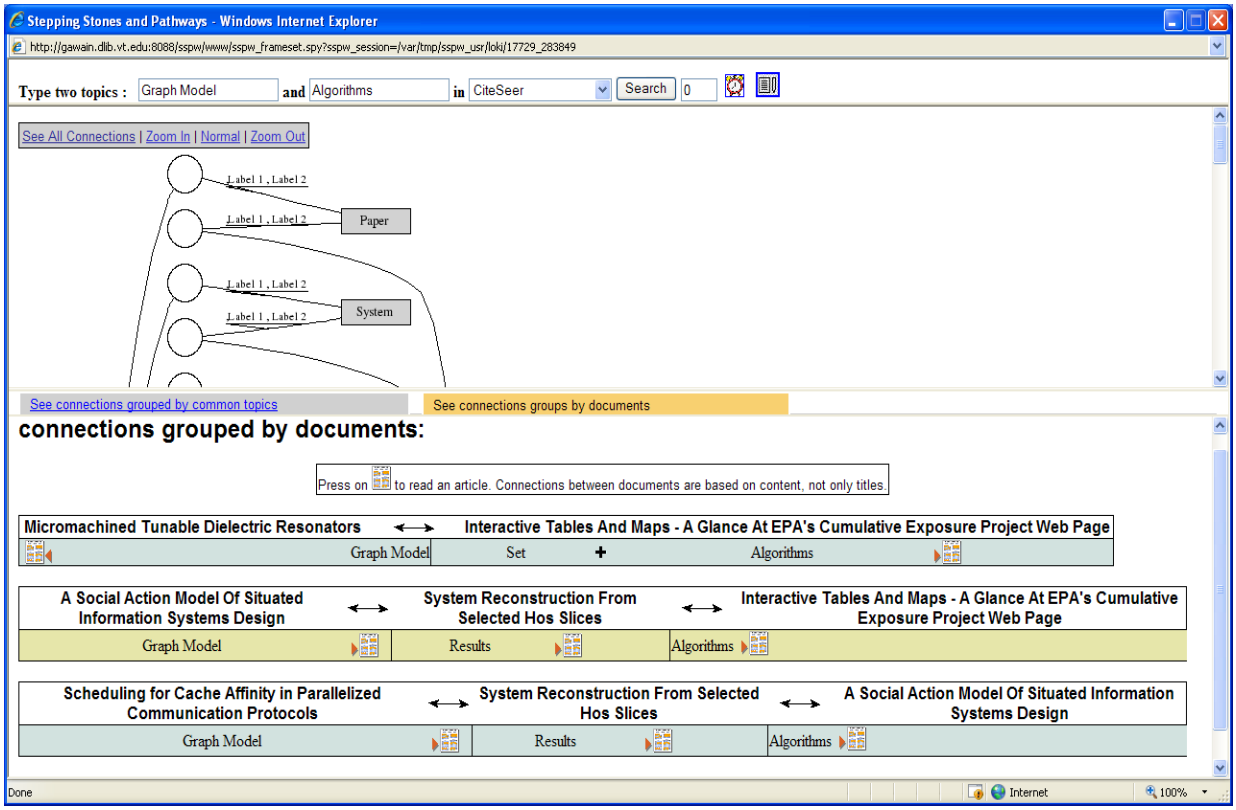

Figure 18: Stepping Stones and Pathways User Interface

The integration part follows a protocol in which messages are sent to the search engine, and the appropriate method gets executed. As an output, this method returns the required information. Detailed information about messages that are sent between the front end and the back end are explained below. The 3 major components, which interact, are given below:

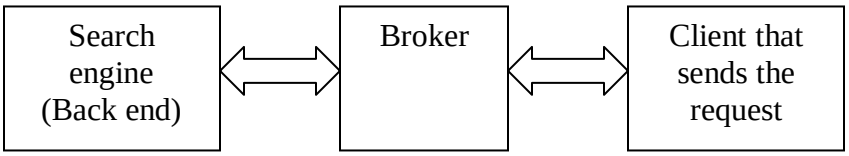

Figure 19: Components of Stepping Stones and Pathways

The client sends the user's query command to the broker. Then the broker sends that command to the back end. It gets executed in the back end which sends a message back to the broker, which in turn is sent back to the client. The set of commands and their interactions are displayed in detail below in the order in which they are sent in the form of a sequence diagram:

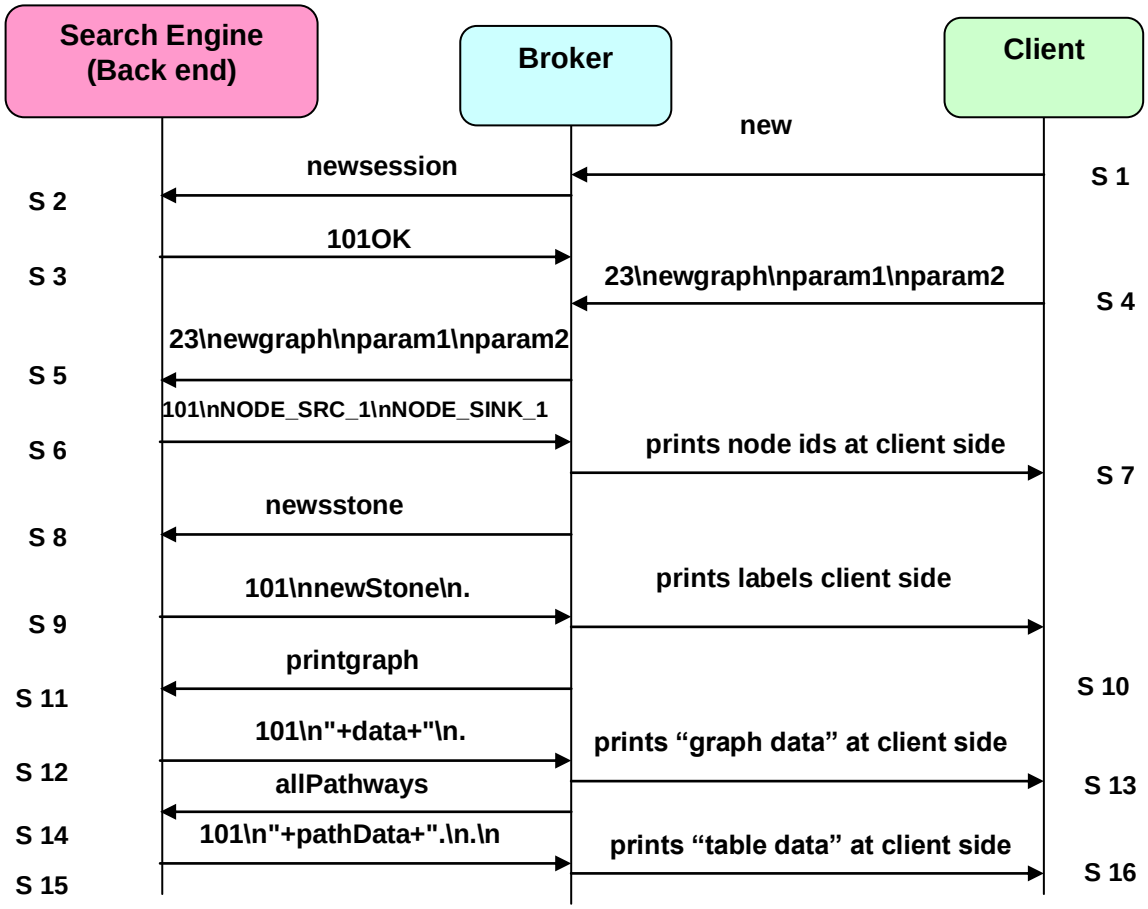

Figure 20 : Sequence Diagram between Client and Search Engine

5.7.1 Sequence of Steps in Client and Back End Interaction

1. **newsession:**

This command is sent to the search engine from the client when the user wants to open a new session between the client and the server. The server is a multithreaded program. It waits for the request on a certain port. Any number of clients can send requests and the server processes these requests.

The search engine sends 101OK as a reply to the client.

2. **newgraph:**

The newgraph command is sent to the search engine along with the two input queries as parameters, which are processed at the back end. The request and the parameters sent is of the form:

23\newgraph\nparam1\nparam2, where
param1 = query1 (for e.g.: “operating systems”)
param2 = query2 (for e.g.: “virtual machines”)

When the search engine receives these parameters, an appropriate method is called with these as input, and query processing starts. When the search engine receives the result, it is sent in the following format:

101\nNODE_SRC_1\nNODE_SINK_1, where

node_src_1 -> is the node id for the end stone 1 (source)
node_sink_1 -> is the node id for the end stone 2 (sink)

3. **newstone:**

The newstone command is sent when the client needs the labels of the stones from the search engine. The back end sends back the output in the following form:

101\nnewStone\n. Where

newstone -> the label which is being sent out and

\n -> New line symbol which is being sent to client

. -> dot represents the end of message or data, which is being sent to the client

4. **printgraph:**

The printgraph command is sent from the client when the client needs to print the UI data in the output screen. As soon as this command is received the back end sends the data to the client in the following format:

101\n"+data+"\n.

The "data" in the above denotes the graph data that is constructed by the back end and sent to the front end for displaying results to the users.

5. **allPathways:**

The allPathways command is sent by the client to the back end when the client needs to see all the documents in the form of a table of data in the browser. The result that is being sent back to the front end is of the form:

101\n"+pathData+".\n.\n

Here "pathData" is the table data that is sent to the front end for displaying retrieved documents information, in the form of the table, to the users.

Chapter 6.Technology

This chapter discusses the various tools and technologies which are used to implement the scalable Stepping Stones and Pathways.

The following table shows the list of tools used for this implementation:

Purpose	Tool	Links
Indexing	Lucene	http://www.apache.org/dyn/closer.cgi/lucene/java/
Searching	Lucene	http://www.apache.org/dyn/closer.cgi/lucene/java/
Matrix creation	Mt4J	http://ressim.berlios.de/
Analyzer	Lucene	http://www.apache.org/dyn/closer.cgi/lucene/java/
Browse Index	Luke	http://www.getopt.org/luke/
Query expansion	LucQE	http://www.apache.org/dyn/closer.cgi/lucene/java/
Graph creation	JGraphT	http://jgrapht.sourceforge.net/
Clustering	Carrort2	http://project.carrot2.org/

Table 2: Tools used in Scalable SSP Implementation

6.1 Lucene

The search engine component of the initial SSP is replaced with Lucene, which is a full-featured text search tool kit that has support for scalability and high performance. Since it is written in Java it can support cross-platform deployment. Lucene, through its API, offers many compelling features. They are classified into the following three major categories:

1. Scalable and High performance Indexing:
    - It is estimated by Apache that Lucene can index about 20 MB of data in 1 minute on a Pentium 1.5 GHz machine.
    - It requires less RAM. One MB of heap is sufficient.

- It has support for incremental indexing.
- It gives good performance.

2. Effective and powerful search algorithms:
    - Ranked searching
    - Support for different query types
        - Phrase queries.
        - Wildcard queries.
        - Proximity queries.
        - Range queries.
    - Multiple index searching with merged results
    - Support for parallel update and search
3. Cross platform support
    - Since Lucene toolkit is Java based, it supports class platform deployment

SSP uses mainly the following API for indexing and querying:

- Analyzer
- Document
- IndexWriter
- IndexReader
- IndexSearcher

6.2 LucQE

Query expansion is the process of adding words to the user's weighted search. The idea of this is to improve precision and recall parameters in the search. LucQE provides a framework that supports query expansion (QE) with the use of Apache Lucene. The use of LucQE in SSP is to get the centroid (top 35 frequently occurring words) for the end stone documents, which could serve as the expanded query to get the set of intermediate documents. These form the potential connection documents between the two input concepts. There are many query expansion techniques available in the IR field and LucQE uses the Rocchio query expansion technique.

6.3 MT4J

The matrix tool kit for Java is a comprehensive collection of matrix data structures. It is designed to be used as a library for developing numerical applications for both small scale and large-scale computations.

In SSP, the documents processed can number in the millions. The citations and co-citations are processed using the MT4J tool kit. This requires using a data structure, which handles a huge data set and that provides storage optimization. MT4J has a storage layout that takes into consideration all the dense and structured sparse matrices. Thus, it is appropriate for processing and storing the citation and co-citation data.

Citations:

Citations are documents or papers which a particular paper refers.

Co-Citations:

Co-citations are documents or papers in which a particular paper is referenced.

Some of the MT4J API calls used in our implementation are:

1. FlexCompRowMatrix(matrixSize,matrixSize)
2. SparseVector(matrixSize)
3. MatrixVectorReader(fileName)
4. FlexCompColMatrix(flexRowMatrix)
5. printMatrixInfo()

6.4 Luke

Lucene is a mature, high performance search tool kit. In the scalable SSP implementation the index is created with millions of documents by using the Lucene search tool kit. To provide a clear picture of the index structure, along with other features like search and analyzers, a handy development and diagnostic tool called Luke is available. It is Lucene's index toolbox. Luke allows the users to perform the following tasks:

1. Browsing by document number, or by term.
2. Document viewing or copying to the clipboard.
3. Retrieving a ranked list of most frequently used terms.
4. Perform searching and browsing by:
    a. Choosing the field to search on.
    b. Choosing the analyzers and performing search. There is support for both built-in and custom analyzers.
5. Identifying the high frequency words in the index by selecting the field and limiting the number of words that the user needs to view.
6. Deleting the documents from the index.
7. Reconstructing the original document fields, editing, and re-inserting in the index.

6.5 JGraphT

The SSP project involves creating graphs with the pathways and the end stone nodes, which are generated by the back end. For this JgraphT is used. It is a free Java graph library that provides mathematical graph-theory objects and algorithms. JGraphT supports various types of graphs:

- Directed and undirected graphs.
- Weighted / unweighted / labeled or any user-defined graphs.
- Graphs with one or more edges like:
    - Simple-graphs.
    - Multigraphs.
    - Pseudographs.
- Graphs that can be controlled. For example, given the choice to keep the graph as unmodifiable - allow modules to provide "read-only" access to internal graphs.
- Listenable graphs - allow external listeners to track modification events.
- All combinations of above mentioned graphs.

JGraphT is designed to be simple and type-safe (via Java 5 generics).

- For example, graph vertices can be of any objects. Also graphs can be created based on the following:

    - Strings
    - URLs
    - XML documents
- Other features offered by JGraphT:
    - Graph visualization
    - Easy extensibility

For the scalable SSP implementation the following classes of JgraphT are used:

- SimpleGraph
- DefaultEdge

6.6 Carrot2 Framework

Carrot2 is an open source tool used for clustering results. The advantage of using this framework is its ability to automatically cluster the result set. The flexible architecture and choice of clustering algorithms makes it very useful in a number of commercial and research applications.

Chapter 7. Experiments

7.1 Summary

The experiments are designed to test the new scalable SSP for various scalability factors like number of documents in the collection, time taken to index and query the documents, and storage requirements. The experiments are divided into two categories: 1. Indexing time and 2. Query time experiments. In both cases there are a number of independent and dependent variables. The variables and their types are listed below:

Independent variables:

a. Size of the data set
b. Query used
c. Initial set of documents indexed

Dependent variables:

a. Time taken for the query to execute over the data set
b. Storage (memory) required on a system (disk space)

After measuring these variables, a run time analysis was performed for these experiments to understand the relationship between the input and time taken for indexing or querying. In some experiments profiling was done to understand the time spent in different modules. The experiments were run under various contexts and that is explained in detail along with each experiment.

In our experiments, we use statistical analysis to make inferences. The number of documents processed is one of the main scalability factors which is tested using the experiments described below. Initially, we take the number of documents and perform the statistical test and make conclusions based on the test. However, there are few cases where we run experiments and make conclusions based on the observations. The detailed tests and analysis for these would be performed later.

Statistical test and analysis for scalability of the new SSP system:

Task:

To prove that the new SSP system can handle larger data set (documents in the order of millions) compared to the old system, which could handle documents only in the order of thousands.

Distribution:

The sample is normally distributed.

Null Hypothesis H_0 :

The percentage of documents that the old system can process is greater than or equal to the percentage of the documents that the new system can process.

Alternate Hypothesis H_1:

The percentage of documents that the old system can process is lesser than the percentage of the documents that the new system can process.

Significance Level:

The significance level used for this experiment is equal to 0.05 (α).

Data Collection:

We have a collection of 700,000 documents. The query is run "N" times on the collection, and N is chosen as 100 in our case.

Assumption:

The probability that there will always be an output is considered as 0.5. This is the population mean.

Mean Values:

Mean value for the old system, M1 = 0

Mean value for the new system, M2 = 1

Two Sample proportion test:

The two sample proportion test is performed to prove the above hypothesis.

The Variables used in the test are given below:

- Number of times the query is run for the old system, n1 = 100
- Number of times the query is run for the new system, n2 = 100
- Number of times we get output using the old system, x1 = 0
- Number of times we get output using the new system, x2 = 100
- Proportion of the number of times the system runs for M1, p1 = 0
- Proportion of the number of times the system runs for M2, p2 = 1

The formula used to compute the p-value is given below:

$$Z = \frac{p_1 - p_2}{\sqrt{p(1-p)(\frac{1}{n_1} + \frac{1}{n_2})}}$$

where,

$$p = \frac{x_1 + x_2}{n_1 + n_2}$$

From the assumption, the value of p is 0.05. Substituting the values of the above variables,

$$Z = \frac{0-1}{0.0707} = -\frac{1}{0.0707} = -14.142$$

Conclusion:

The p-value which is obtained using the above test (-14.14) is less than the α value (0.05). This proves that our null hypothesis, that the old system can process more percentage of documents than the new system is disproved. This proves the alternate hypothesis that the new system can handle documents more than the old system.

7.2 Indexing Time Experiments

The indexing involves processing the XML records and creating an index with the data elements. Experiments were performed for this step to find out the relationship among size of data set, time taken to index, and storage requirements.

The documents are indexed with the following flags for the fields:

Field.Store.YES
Field.Index.TOKENIZED
Field.TermVector.YES

Following are the details of the experiments:

7.2.1 Experiment 1: Indexing Time Analysis

Context:

The index is created with the above mentioned conditions. A different data set size is used for every run and a new index was created during each run. The run time for creating an index for every new data set size is measured in this experiment. The experiment was started with 20,000 records and we continued doubling the number of records to be indexed in every run. The time taken to create the index for each run was measured.

Input: XML records
Output: Index created with the data from XML records

Measure:

We processed various size data sets and analyzed how much time was required to index each set (number of documents).

Variables:

- Independent variable: Size of the data set
- Dependent variable: Time taken to index this data set

Analysis: Time Analysis

Table 3 shows the average time taken for indexing the documents, measured in minutes. The graph corresponding to this is shown in Figure 21, with time in the Y-axis and number of documents in the X-axis. The experiment was repeated for the same input data set for 'N' number of times, N = 5 in our case.

From every run, where the number of records to be indexed is doubled, the time taken to index is also doubled. We plotted the graph for this and observed the nature of the graph. It appears that the time increases linearly with the increase in the size of the data set.

Number of Documents	Indexing time in minutes
40000	5.67
80000	13.98
160000	35.54
320000	87.86
640000	190.91
1620000	482.97

Table 3: Indexing Time Analysis

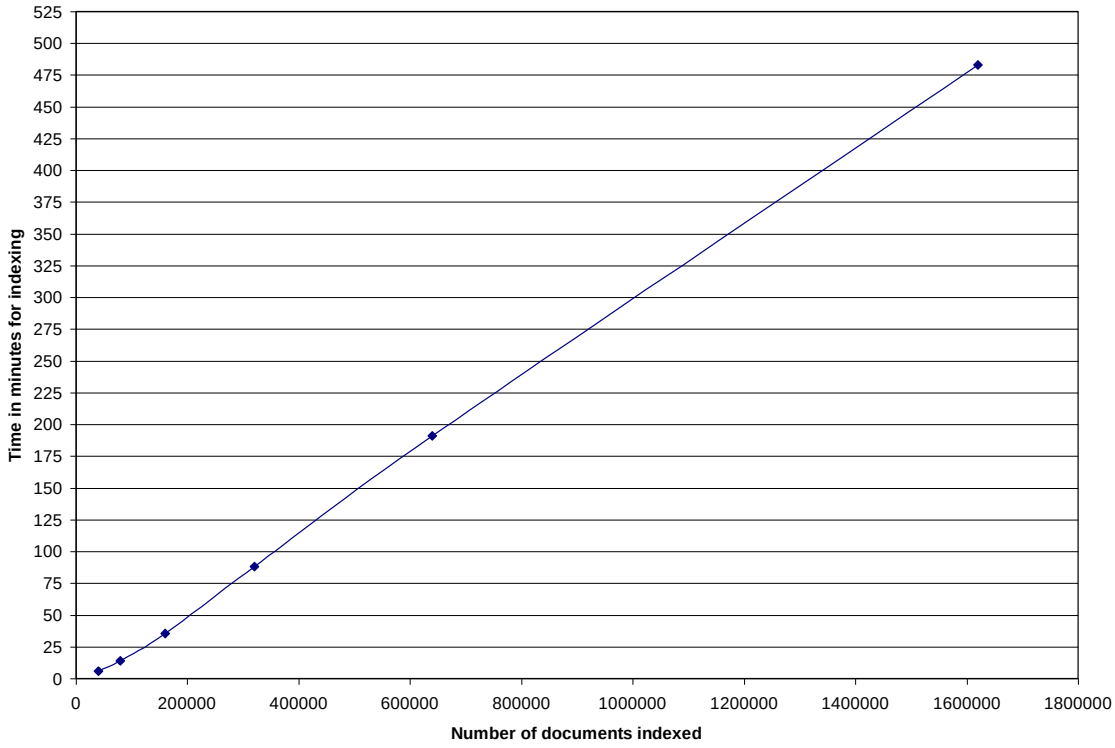

Figure 21: Indexing Time Analysis

7.2.2 Experiment 2: Indexing Space Analysis

Context:

The index was created with the above mentioned conditions. A different data set size was used for every run and a new index was created during each run. The storage requirement for creating the index with each new data set size was measured in this experiment. This experiment was started with 20,000 records and continued doubling the number of records for every run, creating a new index and measuring the space taken.

Input: XML records

Output: Index created with the data from XML records

Measure: We measured the memory (storage) requirement for various size data sets.

Variables:

- Independent variable: Size of the data set
- Dependent variable: Memory (disk space) required for this index

Analysis: Space Analysis

Table 4 shows the storage space taken by the index, measured in megabytes. The graph corresponding to this is shown in Figure 22, with storage space in the Y-axis and number of documents in the X-axis. The experiment was repeated for the same input data set for 'N' number of times, N = 5 in our case.

From every run, we found that when the number of records to be indexed is doubled, the storage space taken by the index is also doubled. We plotted the graph for this and observed that the storage space increases linearly with the increase in the size of the document set.

Number of Documents	Storage requirements in megabytes
20000	33.79
40000	67.71
80000	135.37
160000	274.05
320000	547.78
640000	1079.03

Table 4: Indexing Space Analysis

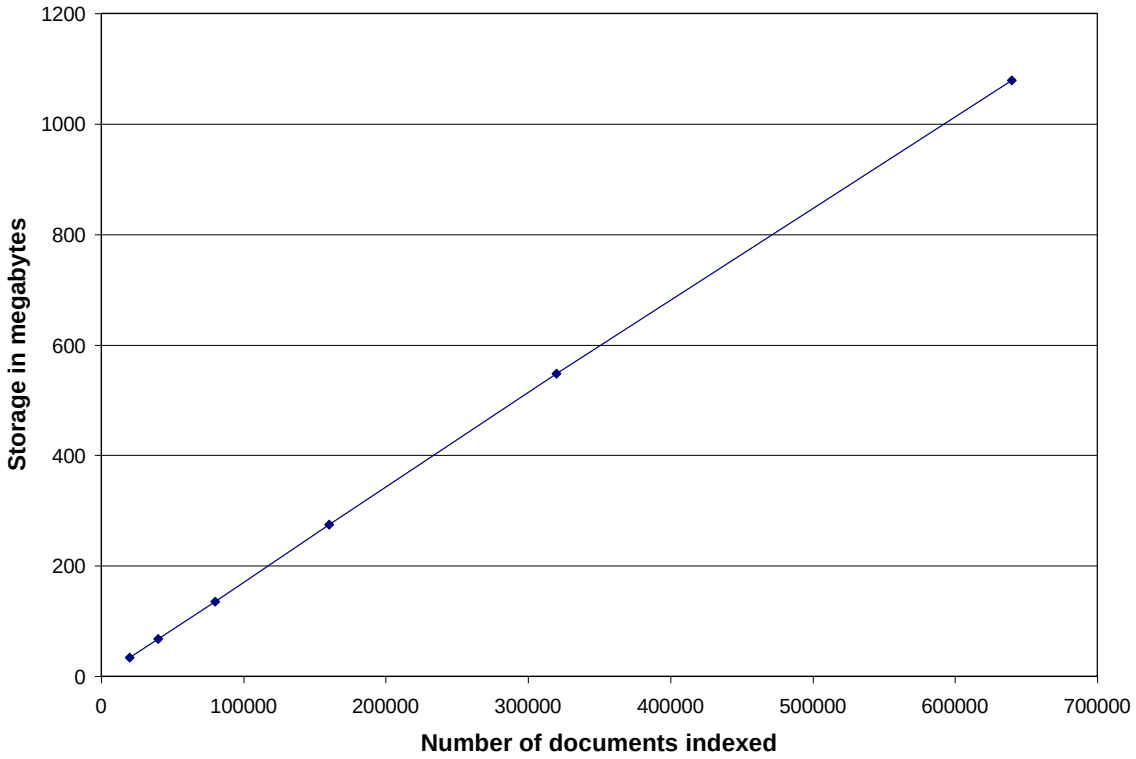

Figure 22: Indexing Space Analysis

7.2.3 Experiment 3: Incremental Indexing Time Analysis

Context:

The index was created initially with a set of records (for example, 20,000 records). To this initial index a new set of records was appended and the time taken to add the new set to the existing index was measured.

Input: XML records

Output: Index created with the data from XML records

Measure:

We measured the incremental indexing time for different data set size (incremental set size)

Variables:

- Independent variable: Size of the incremental data set
- Dependent variable: Time required to incrementally index the new data set.

Analysis: Time analysis

The time taken for indexing the documents incrementally was measured in this experiment. The graph corresponding to this is shown in Figure 23, with time in the Y-axis and number of documents in the X-axis. The experiment is repeated for the same input data set for 'N' number of times, N = 5 in our case.

From every run, we found that when the number of records to be indexed is doubled, the time taken to index is also doubled. We plotted the graph for this and observed that the time increases linearly with the size of the incremental data set.

Base 20,000: The initial set of documents was taken as 20,000 documents and each subsequent run was incrementally doubled in size. The time taken to index the documents was measured.

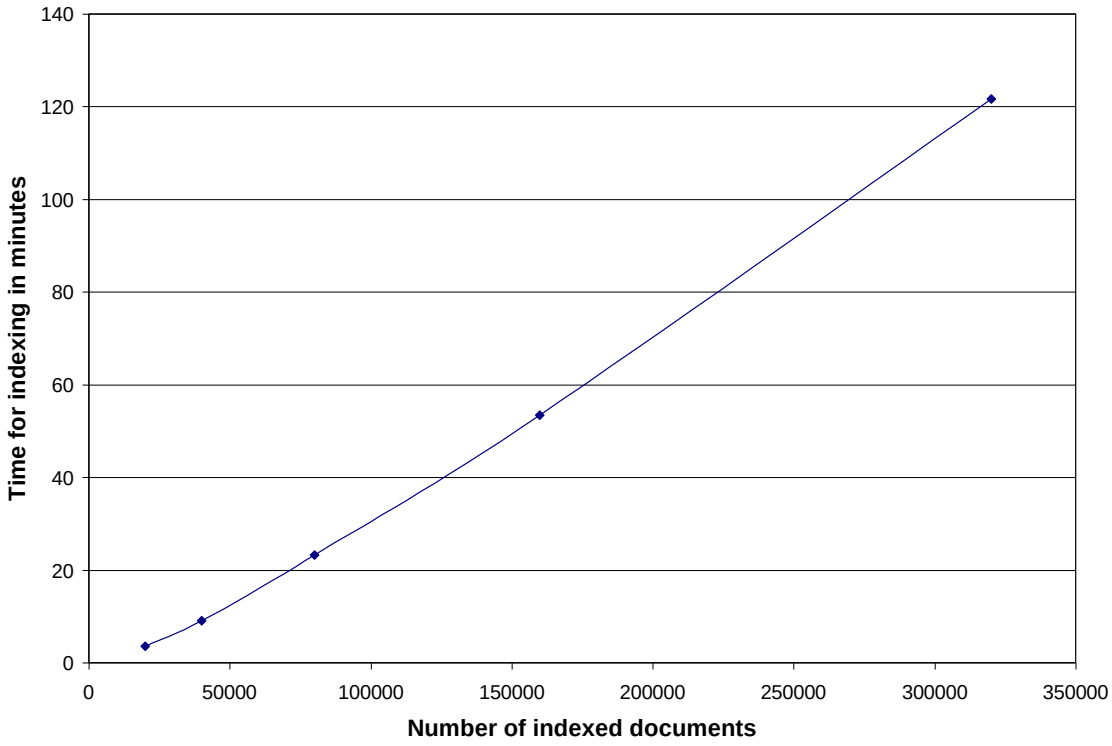

Figure 23: Incremental Indexing with Base 20,000 Documents

Base 40,000: The initial set of documents was taken as 40,000 documents and each subsequent run was incrementally doubled in size. The time taken to index the documents was measured.

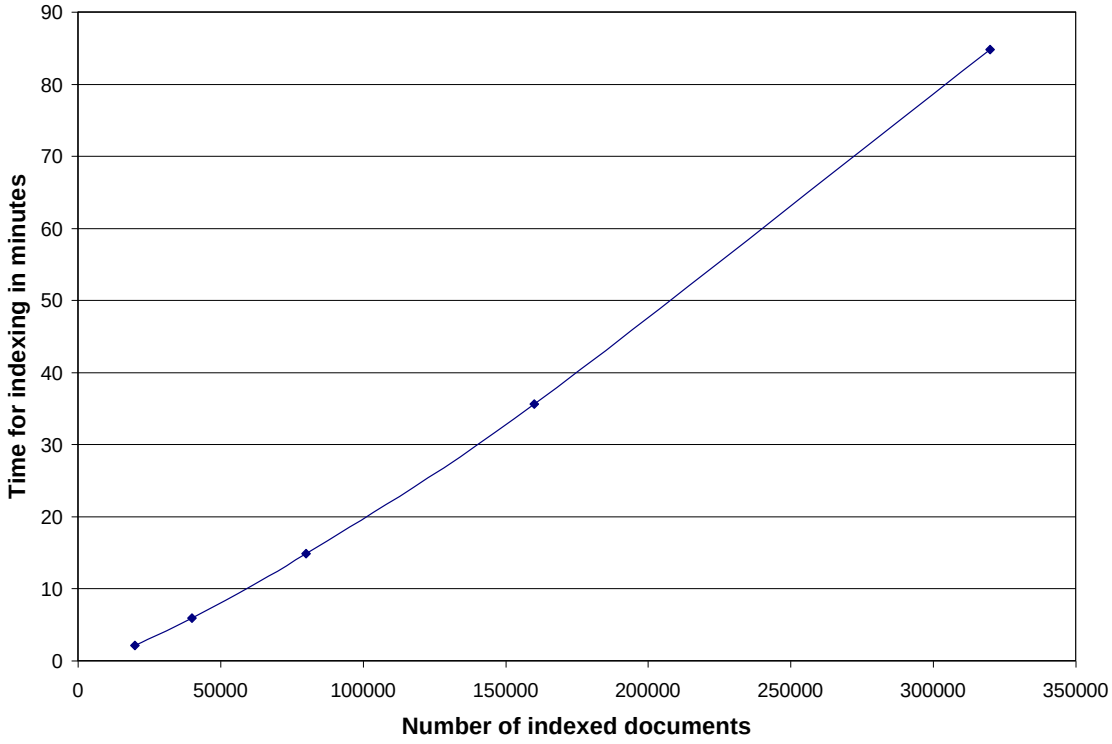

Figure 24: Incremental Indexing with Base 40,000 Documents

Base 80,000: The initial set of documents was taken as 80,000 documents and each subsequent run was incrementally doubled in size. The time taken to index the documents was measured.

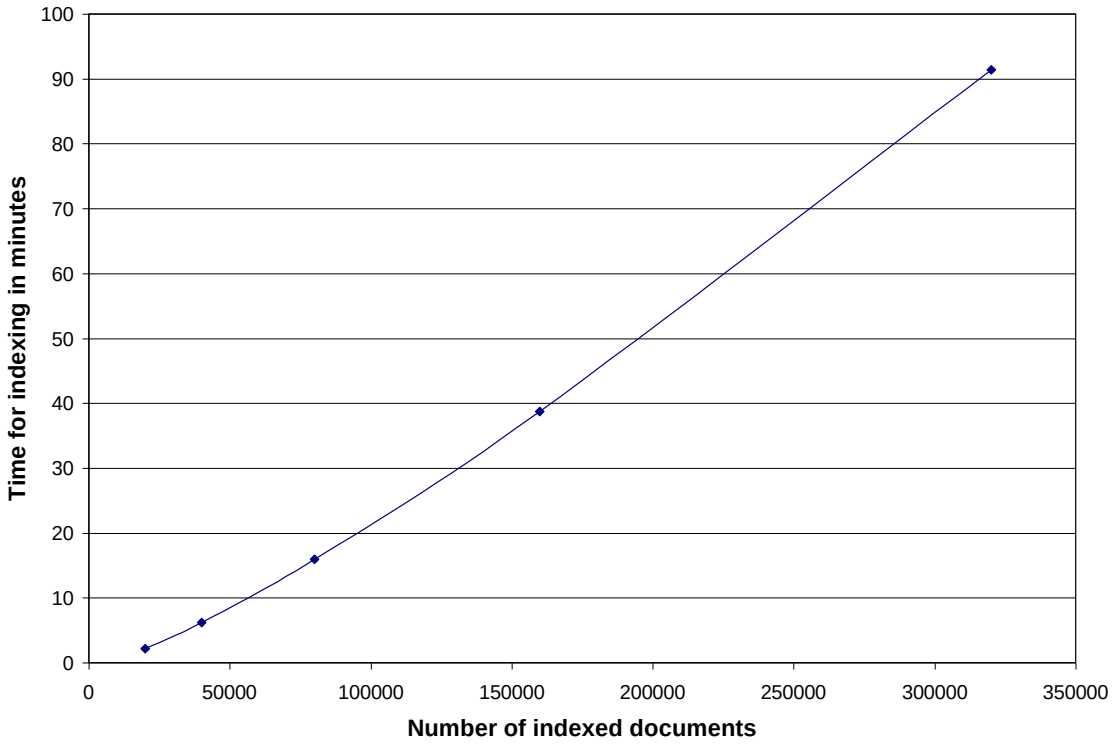

Figure 25: Incremental Indexing with Base 80,000 Documents

Base 160,000: The initial set of documents was taken as 160,000 documents and each subsequent run was incrementally doubled in size. The time taken to index the documents was measured.

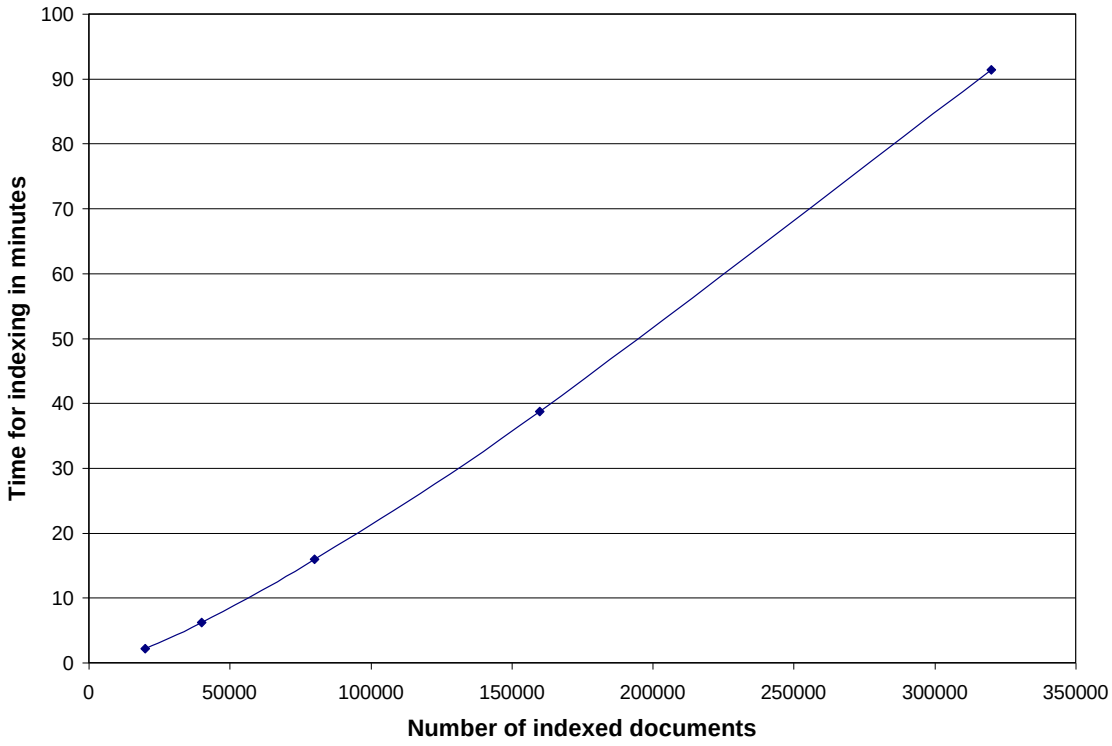

Figure 26: Incremental Indexing with Base 160,000 Documents

Base 320,000 : The initial set of documents was taken as 320,000 documents and each subsequent run was incrementally doubled in size. The time taken to index the documents was measured.

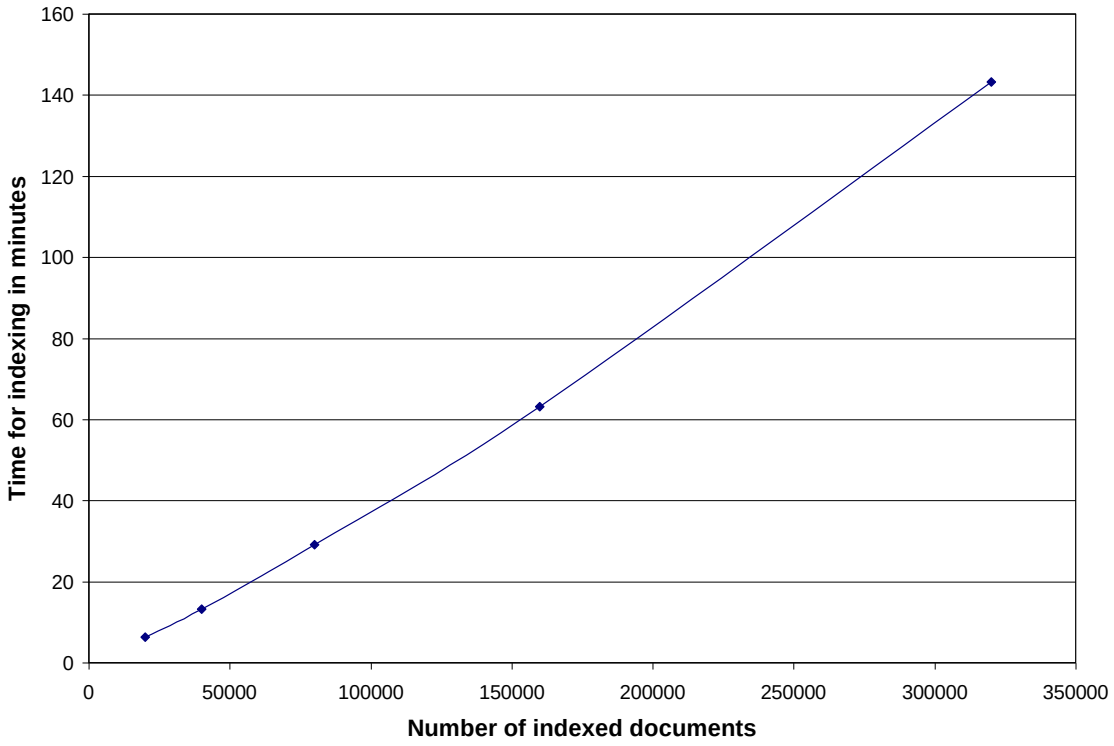

Figure 27: Incremental Indexing with Base 320,000 Documents

7.3 Query Time Experiments

The query time experiments are used to measure the time taken to execute the query with various set up conditions. These are explained in each of the experiments below:

7.3.1 Experiment 1: Effect of Changing Number of End Stone Documents

Input : Queries

Output : Ranked pathways

In this the number of documents in end stones were changed and the time taken to process the input queries was measured.

Variables:

- Independent variable: Number of end stone documents

- Dependent variable: Time taken for the query to execute for different numbers of end stone documents
- Dependent variable: Number of pathways that would result

Constraints:

In this experiment it is imperative that the results are not empty pathways. In order to achieve this a minimum number of end stone documents, intermediate documents, and minimum number of pathways were required.

Profile:

In the query time experiments, the time taken was profiled for each of the following modules:

- End stone creation (each)
- Intermediate document set creation
- Pathways formulation
- Graph creation

Analysis: Time analysis

In this experiment, the time taken to execute the query was measured when the number of documents in the end stone was changed. The graph corresponding to this is shown in Figure 28, with time in the Y-axis and number of end stone documents in the X-axis. The experiment is repeated for the same input data set for 'N' number of times, N = 5 in our case.

From every run, we found that when the number of end stone documents to be retrieved is doubled, the time taken to query that is also doubled. We plotted the graph for this and observed that the time taken for query execution grows linearly with the increase in the number of end stone documents.

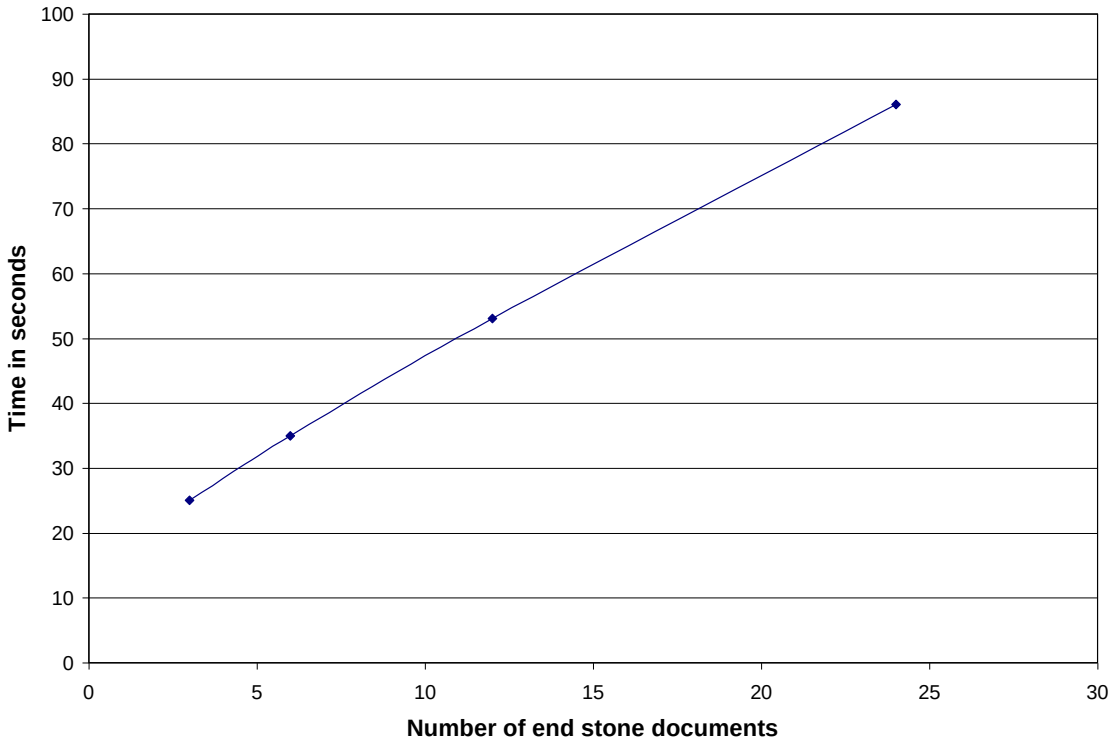

Figure 28: Effect of Changing Number of End Stone Documents

7.3.2 Experiment 2: Effect of Changing Number of Intermediate Documents

Input: Queries

Output: Ranked pathways

In this the number of intermediate documents were changed and the time taken to process the input queries was measured.

Variables:

- Independent variable: Number of intermediate documents
- Dependent variable: Time taken for the query to execute for different numbers of intermediate documents

- Dependent variable: Number of pathways that would result

Constraints:

In this experiment it is imperative that the results are not empty pathways. In order to achieve this a minimum number of end stone documents, intermediate documents, and minimum number of pathways were required.

Profile:

In the query time experiments, the time taken was profiled for each of the following modules:

- End stone creation (each)
- Intermediate document set creation
- Pathways formulation
- Graph creation

Analysis: Time analysis

In this experiment, the time taken to execute the query was measured when the number of documents in the intermediate document set was changed. The graph corresponding to this is shown in Figure 29, with time in the Y-axis and number documents in intermediate set in the X-axis. The experiment is repeated for the same input data set for 'N' number of times, N = 5 in our case.

From every run, we found that when the number of intermediate documents to be retrieved is doubled, the time taken to query that is also doubled. We plotted the graph for this and observed that the time taken for query execution grows linearly with the increase in the number documents in the intermediate document set.

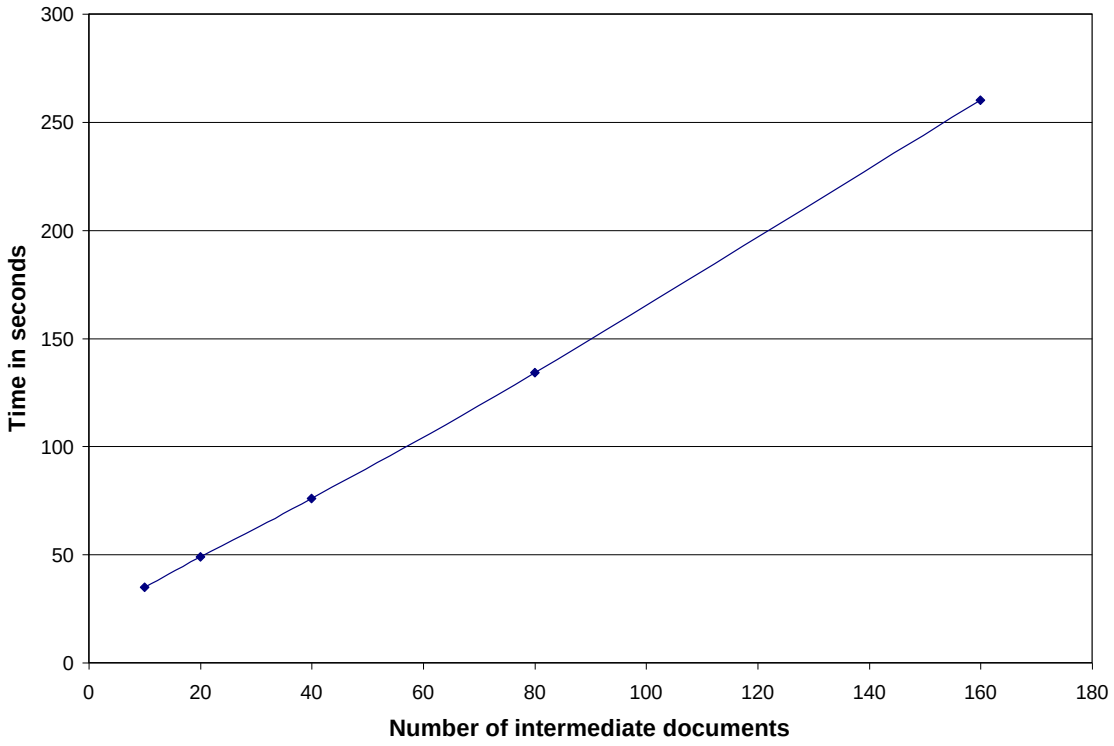

Figure 29: Effect of Changing Number of Intermediate Documents

7.3.3 Experiment 3: Effect of Changing Number of Pathways

Input: Queries

Output: Ranked pathways

In this the number of pathways were changed and the time taken to process the input queries was measured.

Variables:

- Independent variable: Number of intermediate documents
- Dependent variable: Time taken for the query to execute for different numbers of pathways
- Dependent variable: Number of pathways that would result

Constraints:

In this experiment it is imperative that the results are not empty pathways. In order to achieve this a minimum number of end stone documents, intermediate documents, and minimum number of pathways were required.

Profile:

In the query time experiments, the time taken was profiled for each of the following modules:

- End stone creation (each)
- Intermediate document set creation
- Pathways formulation
- Graph creation

Analysis: Time analysis

In this experiment, the time taken for the query to execute was measured when the number of pathways was changed. The graph corresponding to this is shown in Figure 30, with time in the Y-axis, and number of pathways in the X-axis. The experiment is repeated for the same input data set for 'N' number of times, N = 5 in our case.

From every run, we found that when the number of pathways to be created is doubled, the time taken to formulate the pathways is also doubled. We plotted the graph for this and observed that the time taken for query execution grows linearly with the increase in the number of pathways.

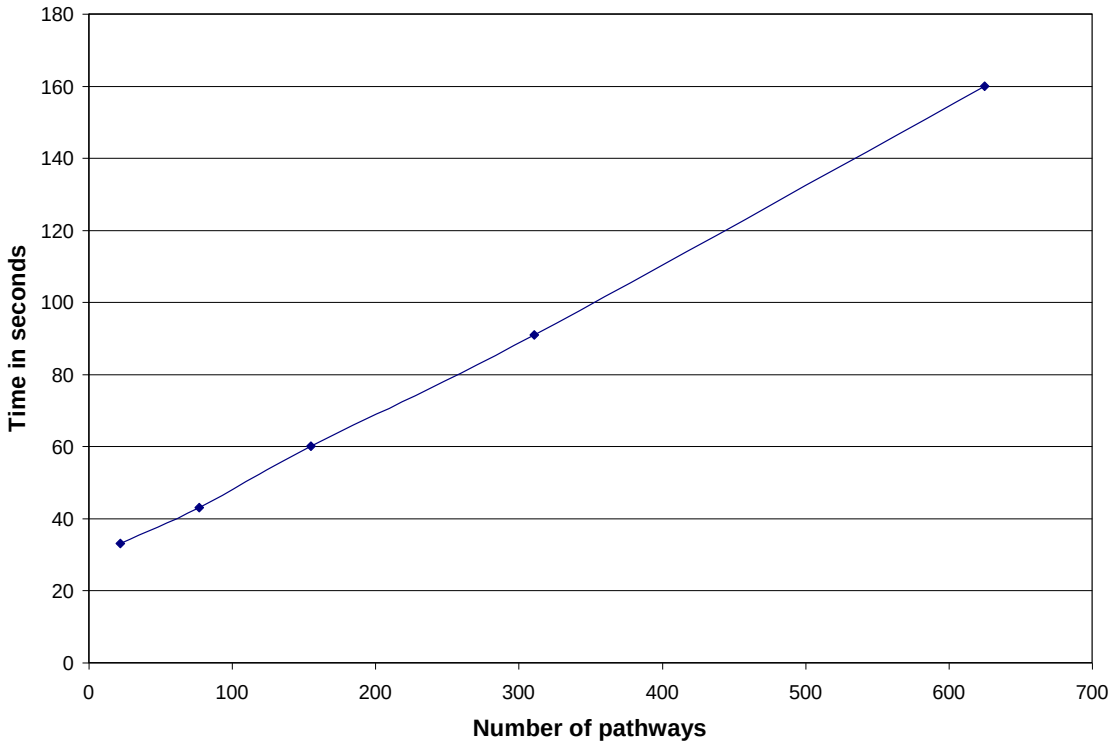

Figure 30: Effect of Changing Number of Pathways

The experiments for the indexing and querying are designed to assess some key factors like:

- Number of documents processed with the new implementation.
- Time taken for query execution for different data set sizes.

The scalability of the system is analyzed with the experiments performed for the above factors. From the analysis, we conclude that the new scalable SSP implementation has shown notable scalability improvements.

Chapter 8. Deploying Scalable SSP

This chapter discusses the library files, configuration by using property files, and the source structure, packaging and deployment.

8.1 Library Files

The scalable Stepping Stones and Pathways uses library files for indexing, query expansion, matrix creation, clustering, and graph creation. In case of our implementation these files are organized under the **/home/user/SSP/lib** directory structure on the server **gawain.dlib.vt.edu**. The code can be deployed on any machine with the same directory structure. The *classpath* should be set to refer to this, so that the source code has reference to these library files.

The list of library files and their purpose is given in the table below:

Library Files	**Jars/Tools**
Lucene indexing and searching	lucene-core-2.0.0.jar
Matrix toolkit for Java	mtj.jar
Graph	jgrapht-jdk1.5.jar
Logging information	log4j-1.2.13.jar
Lucene query expansion	com.zip
Clustering	They are nearly 30 jar files as a part of the clustering library and they are available under /home/user/SSP/clustering/

Table 5: Library Files used in Scalable SSP Implementation

8.2 Configuration Files

The current SSP implementation uses configuration files that includes the key-value pairs for some of the customizable parameters. Currently, there are three types of configurations as listed below:

1. Logging parameters
2. Query expansion parameters
3. Search engine parameters

Each of these are saved as separate property files with the **.prop** extension. They are processed using the *Property* API from the standard Java 1.5.0_08 development toolkit. Following is the list of configuration files used in this implementation:

Logging parameters	**/home/user/SSP/trunks/conf/log.prop**
Query expansion parameters	**/home/ user /SSP/trunks/conf/qe.prop**
Search parameters	**/home/ user /SSP/trunks/conf/searchengine.prop**

Table 6: Configuration Files used in Scalable SSP Implementation

8.3 Source Files

Our source code for this implementation is available in the server **gawain.dlib.vt.edu,** under the **/home/user/SSP/src** directory with appropriate package structure. The main directory under which the main search application class is available is /src/edu/vt/searchengine/. The source code can be deployed on any machine where there is support for Java, the library files, and the configuration files. These files should be deployed in the appropriate directory structure relative to the source code deployment. The *classpath* environment variable is used to resolve the references to the library and other class files.

8.4 Issues

Following are the issues encountered during implementation of the new system. The issues were addressed by taking appropriate steps. The list of issues and the fixes are explained in detail below.

XML dump file data issues:

1. The metadata dump files have extra double quotes. The XML parser cannot process XML data which are not well formed. So the presence of double quote causes exceptions while parsing data. These records are identified and the erroneous files are marked so they can be reported to CiteSeer for fixing this data issue.

2. The presence of control characters like ^Z, ^H, and ^@ causes exception when the XML data is parsed. This is a file format issue. The XML parser causes the following error message due to this issue: *Unicode character set not supported.* The records having this issue are identified and the erroneous files are marked so they can be reported to CiteSeer personnel for fixing this data issue.

3. When the document is in unrecognizable language, there may not be support for those Unicode characters, which causes parsing issue. The records having this issue are identified and the erroneous files are marked so they can be reported to CiteSeer personnel for fixing this data issue.

4. Some documents do not have a value for "description" which is semantically not a valid case, because every document should have all three field values: Identifier, Title, and Description. When such data errors are encountered they are logged and reported to the CiteSeer staff. These issues should be fixed by CiteSeer staff so that the metadata have all the necessary elements and their corresponding values.

5. Reference identifiers, which are quoted in any document, may not be present. This is a missing document scenario and causes issues during the resolution of references. These errors are identified, logged, and reported.

Chapter 9. Conclusion and Future Work

This chapter summarizes the findings in the scalable SSP implementation. The future work section describes some topics that are not covered as a part of the scalable SSP implementation but worth pursuing as further research.

9.1 Conclusion

It is possible to achieve the scalability goal in the Stepping Stones and Pathways system by replacing the back end with scalable components. It is shown to be possible to process records numbering in the millions using the new SSP system, and it was verified using the CiteSeer data collection. The system can deal with the growth in CiteSeer data by indexing the new records incrementally.

The experiments were performed with the new solution and scalability factors like indexing time, searching time, and storage were measured. The results have proved that the new scalable system can work well for larger data set. Profiling was done to measure time spent in various runtime steps. We analyzed the profiled time and inferred the following:

- The query execution time increases when we increase the number of relevant documents to be retrieved. This happens for both the initial query execution where the end stone documents are created, and for the expanded query execution, where the intermediate document set is created.
- With the increase in the number of documents in the intermediate document set, the time taken to perform the N X N document comparison also increases. One reason could be due to the time spent in retrieving the actual document from the index, since the intermediate document set only has the document identifiers. In order to perform document comparisons, we need to retrieve detailed information from these documents. Another reason could be due to the time taken to perform similarity value calculations using the vector space cosine similarity approach. One can perform detailed profiling to find more information for these steps.
- The pathways creation time increases with the increase in the number of documents.

The profiling helped in understanding the maximum time spent in retrieval which could be addressed by parallel processing, caching, and duplicating the index.

9.2 Contributions of this Work

- A new scalable SSP engine that can handle larger data sets
- A thorough understanding of one new test collection and different ways to measure the scalability factors using this collection with the new system
- Introduction to various scalable IR tools and technologies that can be used to build an effective IR system

9.3 Future Work

In this work the primary goals have been accomplished, but there is scope for improvement related to performance, user interface, and clustering. There are four main areas that can be worked on to make further improvements in the current scalable SSP system. Following is the list of areas for further work:

- Improvements in the user interface
- Parallelizing the steps in the new framework
- Performance improvements over existing work
- Improvements in current clustering

9.3.1 User Interface Improvements

The current SSP results' output format is much different from the classical search output format. In the classic search the output is displayed as a ranked set of documents. However, the output in SSP contains a list of document chains displayed as two separate parts. The first part is a graph showing document chains in the form of nodes and edges. The second part is the table data format. This new type of query output requires new users to undergo training sessions to interpret such a result representation.

Also in this type of query output methodology, there is a limitation on the number of documents which are displayed in the browser. In the graphical layout where there are nodes and links there is so much white space, which restricts displaying more results. In

the case of Omnipelagos search system, the output is displayed as a group of connection documents ranked in the order of relevance. This closely resembles the classic search output. This can be adopted in the SSP system, so that it reduces the learning curve involved in interpreting the SSP graph output methodology.

9.3.2 Parallelization in Scalable SSP

In the current scalable SSP algorithm all the steps are executed in sequence. However, there is a scope to execute some of these steps in parallel. The list of steps which can be executed in parallel are : 1. end stone creation, 2. query expansion, and 3. intermediate document set comparison. To understand thoroughly about the runtime one needs to profile the time taken to execute a query from the time it is input by the user.

9.3.3 Performance Improvements

The search applications used in commercial organizations deal with huge document sets. The key idea in using a bigger collection is to improve the retrieval coverage. However, the bigger collections could cause an increase in the query processing time and this is a big trade-off in search. SSP query processing primarily involves tasks like searching the index for the matching documents, performing document-document similarity calculation, and creating pathways from the set of relevant documents in end stones and the intermediate set.

The time taken for query processing varies depending on factors like number of TopDocs retrieved for initial user query, TopDocs retrieved for expanded query, and number of pairwise similarity calculations in the intermediate document set. The query time also depends on index type such as in-memory or file system index. In the scalable SSP implementation a file system index is used which could cause performance trade-offs due to the time taken to retrieve the documents from secondary memory. The other step which might involve more time is the N X N document similarity calculation in the intermediate document set. There is a scope for improving query time by using parallelizing techniques like:

- Parallel execution of the two initial input queries
- Parallel execution of the expanded queries

- Document-Document similarity calculation by using multiple index copies

To accomplish the above, the following steps need to be performed:

1. Profile the time for executing different runtime steps in SSP
2. Analyze and list various steps that could be run in parallel
3. Design different approaches to achieve the parallelization:
   a. Use of shared memory multiprocessors (using MPI for Java)

      OR

   b. Use of clusters (with sockets if the operating system does not schedule tasks to all cluster processors)
4. Implement the new parallel approaches designed
5. Benchmark the SSP system. Measure time using serial (current scalable SSP system) and parallel architectures
6. Analyze the results

9.3.4 New Data Set

The current CiteSeer data set has support for 700,000 documents and is used for current testing. New data sets from various domains, which have different collection sizes, can be used to test the system further. This can help analyze the behavior of the system across various domains. Currently the system can process documents numbering in the millions, but this can be increased further to test collections with billions of documents. This might require a huge collection which can be indexed and tested.

9.3.5 Improvements in the Current Clustering Mechanism

The contemporary clustering mechanism cannot handle huge clustering. The current scalable SSP system uses dynamic clustering, where the intermediate set of documents are clustered as opposed to clustering the whole CiteSeer collection. Hence, the number of documents clustered in the new scalable implementation is a subset of the whole CiteSeer collection. So in the current system dynamic clustering is performed whereas offline clustering is the ideal approach. The clustering could improve the retrieval effectiveness to a great extent.

References

[1] Fernando Das-Neves, Edward A. Fox, Xiaoyan Yu: 'Connecting topics in document collections with stepping stones and pathways', Proceedings of the 14th ACM international conference on information and knowledge management, 2005, 91-98, Bremen, Germany.

[2] Deept Kumar, Naren Ramakrishnan, Richard F. Helm, Malcolm Potts 'Algorithms for storytelling', Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining 2008, Volume 20, 604-610, Philadelphia, USA.

[3] '**http://www.omnipelagos.com/index'**, 2005.

[4] Gresock Joe: 'Finding Combinatorial Connections between Concepts in the Biomedical Literature', Virginia Polytechnic Institute and State University, 2007, Blacksburg , USA.

[5] Helen L. Johnson, K. Bretonnel Cohen, William A. Baumgartner Jr., Zhiyong Lu, Michael Bada, Todd Kester, Hyunmin Kim, Lawrence Hunter: 'Evaluation of lexical methods for detecting relationships between concepts from multiple ontologies', Pacific Symposium on Biocomputing, 2006, 28-39.

[6] Lossau, N.: 'Search Engine Technology and Digital Libraries', D-Lib Magazine, June 2004, Volume 10, http://www.dlib.org/dlib/june04/lossau/06lossau.html.

[7] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, John W. Tukey: 'Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections', Proceedings of the 15th Annual International ACM/SIGIR, 1992, 318-329, Copenhagen, Denmark.

[8] Sanjay Ghemawat, Jeffery Dean: 'MapReduce: Simplified Data Processing on Large Clusters', Proceedings of the Sixth Symposium on Operating System Design and Implementation, 2004, Volume 6, 10, San Francisco, USA.

[9] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd: 'The PageRank Citation Ranking: Bringing Order to the Web', Stanford Digital Library Technologies Project, 1999, http://dbpubs.stanford.edu:8090/pub/1999-66.

[10] Sergey Brin, Lawrence Page.: 'The Anatomy of a Large-Scale Hypertextual Web Search Engine', Proceedings of the seventh international conference on World Wide Web 7, 1998, Volume 30, 107-117 , Amsterdam, The Netherlands.

[11] Erik Hatcher, Otis Gospodnetic: 'Lucene in Action', Manning Publications Co., 2004, 456 pages, http://www.manning.com/hatcher2/

[12] Jose E. Moreira, Maged M. Michael, Dilma Da Silva, Doron Shiloach, Parijat Dube, Li Zhang 'Scalability of the Nutch search engine', Proceedings of the 21st annual international conference on Supercomputing 2007, 3-12, Seattle, Washington.

[13] Michael M, Jose. E. Moreira, Shiloach D, Wisniewski R.W.: 'Scale-up x Scale-out: A Case Study using Nutch/Lucene', Parallel and Distributed Processing Symposium, 2007, 1-8, Long Beach, California.

[14] Jeffrey Dean, Monika R. Henzinger: 'Finding related pages in the World Wide Web', 1999, 31, 1467-1479, Amsterdam, The Netherlands

[15] Wenjun Sun, Kevin Lu: 'Parallel Query Processing Algorithms for Semi-structured Data', Proceedings of the 14th International Conference on Advanced Information Systems Engineering, 2002, 770-773, Toronto, Canada.

[16] C. Lee Giles, Kurt D. Bollaker, and Steve Lawrence: 'Citeseer: An Automatic Citation Indexing System', Proceedings of Digital Libraries'98, 1998, 89-98, Pennsylvania, USA.

[17] Don R. Swanson, Neil R. Smalheiser, A. Bookstein: 'Information Discovery from Complementary Literatures: Categorizing Viruses as Potential Weapons', Journal of the American Society for Information Science and Technology, 2001, 52, 797-812

[18] Google: 'Enterprise Search Evaluation Guide', 2006, http://www.safira.pt/NR/rdonlyres/C506ED0A-301C-40E9-8E41-57A2E8ED3C20/97/EnterpriseSearchEvaluationGuide.pdf

[19] Taher Haveliwala, Aristides Gionis, Dan Klein, Piotr Indyk: 'Similarity Search on the Web: Evaluation and Scalability Considerations', Technical Report, 2001, Stanford University, http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2001-8&format=pdf&compression=

[20] Burgin, R.: 'The retrieval effectiveness of five clustering algorithms as a function of indexing exhaustivity', Journal of the American Society for Information Science, 1995, 46, 562-572.

[21] Jan O. Pedersen, Marti A. Hearst: 'Reexamining the cluster hypothesis: Scatter/Gather on retrieval results', Proceedings of ACM SIGIR, 1996, 76-84, Zurich, Switzerland.

[22] George Karpis, Eui-Hong Han, Vipin Kumar, 'Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling', University of Minnesota - Computer Science and Engineering, 1999, 32, 68-75, Technical Report # 99-007, Minneapolis, USA.

[23] Eugene Garfield: ‘Citation Analysis as a Tool in Journal Evaluation’, 1972, 1, 527-544, http://www.garfield.library.upenn.edu/essays/V1p527y1962-73.pdf

[24] Edward A. Fox, Khoushik, Joseph Shaw, Russell Rao, Durgesh: ‘Combining the evidence from Multiple Searches’, Information Processing and Management: an International Journal, 1992, 31 (3), 431 - 448, Gaithersburg, Maryland.

[25] ‘Lucene Applications’, **http://wiki.apache.org/lucene-java/PoweredBy**, 2005

[26] Edgar Meij, Maarten de Rijke: ‘Deploying Lucene on the Grid’, Proceedings SIGIR 2006 workshop on Open Source Information Retrieval (OSIR2006), http://en.scientificcommons.org/21615939 , 2006, Amsterdam, The Netherlands.