APPROVAL SHEET

Title of Thesis: Full Text Search based on Lucene and Cloud Adaptability Review

Name of Candidate: Pavan Kumar Hanumantharaya
Master of Science, 2018

Thesis and Abstract Approved: 

Dr. Charles Nicholas
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: November 27, 2018

ABSTRACT

Title of Thesis: Full Text Search based on Lucene and Cloud Adaptability Review

Pavan Kumar Hanumantharaya, Master of Science, 2018

Thesis directed by: Dr. Charles Nicholas,
Professor
Department of Computer Science and
Electrical Engineering

We implement a serial indexing approach using Lucene search engine to find the free text in a relational database. We propose to analyze and compare the results obtained using the Lucene search engine against the conventional MySQL query functionality. Today, many search engines combine various retrieval approaches and indexing to obtain the desired query operations. In modern age, the amount of efficiency in retrieval of all related information is of high importance. Lucene has capability to scale millions of pages and records in short span of time. We analyze the inverted indexing methodology to obtain the related information in a huge set of structured database and discuss the effectiveness of performing such operations. The amount of time required to search the indexed storage is very less compared to the time taken by a conventional computer to query the database. In the end, we review the different mechanisms applicable for using lucene to perform such operations on cloud based storage.

Full Text Search based on Lucene and Cloud Adaptability Review

by

Pavan Kumar Hanumantharaya

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science
2018

I dedicate this work to my mom and brother.

ACKNOWLEDGMENTS

I would like to first express my deepest gratitude to my thesis advisor, Dr. Charles Nicholas for guiding me through my masters research. I am gratefully indebted to his invaluable advice, understanding, patience and motivation for this thesis. It has been a great experience and huge learning curve for me to work under his guidance.

I am also grateful to the GANG of UMBC for continuously supporting in the completion of my thesis. They were the sunshine of encouragement in all stages of my masters degree at UMBC. Finally, I would like to thank my mother, Doddammani, brother Ganesh for being my backbone and motivating me to pursue my dream of studying masters. I dedicate this to them and could not have achieved without them.

Thank You!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

In the past decade due to vast amount of data the storage mechanism of information has changed rapidly. In the current world scenario, many real time applications are storing data from structured(Relational and organized data) to free text information(unstructured data) (22). The increase in unstructured data means the research to find an efficient and effective methods to handle both structured and unstructured data has increased. The significance to find the right combinations to handle the unison of both the data is very high. The traditional relational databases and modern search engines based on full text search have distinct and unique features offering to the developers. But at the same time both have overlapping capabilities which can be combined for better results.

Both structured and unstructured data has very sophisticated tools and methodologies to extract relevant required information for the user. They provide rich tools for ordering and viewing of results based on the user conditions or criteria. Full text search methodology is very useful in case of high volumes of data which cannot be related to each other. It provides very fast methodologies to search for any word or combinations of words in that heap of unstructured data. In case of structured data, relational databases plays a vital role if the requirement is to update or modify any certain type of information. Relational

database is very easy, quick and secure if the functionality is to update information of specific table content.

In real world, it is very important to have both the capabilities and combine functionalities of each other for better application with rich advantages. We can observe that not all fields in a relational database table is single word. For example : In a customer table we can have customer complaint which is completely free text and unstructured. In this case, the fusion of relational database and Lucene pays huge dividends to the user. Hence most of the modern day applications rely on both the technologies to serve the user in the best possible way.

Full Text Search(FTS) increases the relevance of search result by very good margin. It's a pattern matching technique which allows the user to have freedom with search criteria. Currently, every website and web applications uses full text search technique to retrieve the data. Even Google search engine has its own implementation of Full Text indexing for retrieving documents related to the search criteria (32). FTS provides user with a lot of relevant data to the search criteria then performing exact string matching search. For example, if the user searches for Bad and worse in customer complaint database then FTS gives all the documents or informations related to both Bad or Worse. It gives combination of results and also documents related to individual words (14). The traditional MySQL systems tends to provide very less efficiency in this aspect and underperforms in case of large datasets (14). Usually the requests from MySQL retrieves the exact text matching documents rather than retrieving the relevant documents to the search criteria.

FTS search capability in searching results among vast volumes of unstructured data is very rich and provides flexible options for improvisation. Its supports different varia-

tions of searching like basic keyword searching, pseudo natural language processing, use of Boolean operators, proximity operations, Internet style increment, find-similar and much more (16). They excel at categorizing the data based on specific value of certain specified fields. The documents generated by FTS are typically of same type or structure. The documents can contain free form text fields or even non-text data and also combination of constrained text data. Each record or document generated is simply a concatenation of relevant fields of data. In general, the given search word is run against a single field or combination of fields depending on the document structure. Full text search engines generally uses this view of the document to perform indexing or searching on the generated documents. FTS mainly involves 3 different kind of operations: Data gathering, Data Indexing and Searching. FTS analyses every word of the document, creates indexes for the gathered information and uses this generated index for searching the user input entered in the application. It has high speed search, steady performance and favourable infrastructure framework.

Most common type of indexing employed by most of the FTS engines is Inverted indexing. Inverted index employs storing the mappings of words to its location in the database file. It enables fast full text search with an added increased processing capability when the document is added to the database (31). A word level inverted index allows phrase searches but occupies substantial amount of space and memory to complete the operation. In inverted index, it inverts a page centric data structure to word centric data structure. It uses a structure which is designed to enable very fast and efficient full text searches in the database. It generally consists of list of unique words present in the entire document. Therefore it contains the location of all documents where the respective word appears. Inverted index is also the best and most widely used mechanism for supporting adhoc text search. It generally has following functionalities : Fetching the document, Removing the stop words,

Stem to the root word, Record document identification numbers, Merge and store the terms.

In modern full text retrieval systems, inverted index provides improved time and space efficiency along with appropriate maintenance scheme for huge amount of information management. Inverted index provides very fast query processing results when combined with efficient information retrieval systems (21). Many research on inverted index proposes a time and space efficient random access block inverted index and dynamic maintenance scheme for increasing the retrieval results in large scale information storage systems (21). The size of the index created varies depending on document id and the count of it occurrences in different records. A efficient and systematically structured inverted index size will be as less as one eight of the original document.

In the context of the modern information flow, FTS has huge number of advantages over the traditional relational database system. The major and prime advantage of FTS is the ability to handle efficient and optimized retrieval mechanism in large amount of unstructured textual data. It is specifically optimized to handle multiple valid and invalid characters. In particular, when the fields are generally represented in many un-normalized form. It provides granular index structure supporting fast access to the indexed document while performing search operations. It supports numerous query operations famously named as Fuzzy matching algorithms. It provides query operations such as fast wildcard matches, word densities, proximity, language stemming, statistically based word similarities, thesaurus, soundex and much more. It provides fusion and composite search support to search both free text data and traditional structured field data. Its rich in providing hybrid search operation. It permits fine tuning of search results by introducing relevancy weighting to the similar documents (15).

FTS systems returns list of meta-data representing documents after processing the query unlike returning list of rows in a table structure. A result set record in relational database is same as document in FTS. It supports nested queries along with the combination of different query operators. FTS additionally retrieves documents weighted using the relevancy score along with exact match of the query documents(15). The weight is usually calculated using many proprietary algorithms and sort techniques. Lucene usually uses the Boolean model to calculate the weight and practical scoring functionality to calculate relevance. It uses different usage patterns to query the resulting output documents. It generally sends shorter initial queries and redefine them later to find the related documents. In FTS all the grouped items are indexed into the same collection. The documents in FTS are identified using document identification numbers. It acts like a primary key while performing indexing and searching operations (15).

Lucene is a full-text search library in Java used to add simple and efficient search functionality to an application or website or local database. It is a free, open-source information retrieval software library originally created by Doug Cutting and released under Apache Software Foundation. It is capable of high performance indexing and virtually search any kind of text in the indexed data. It primarily focuses on Information Retrieval as compared to Huge data Management. It is scalable and can also be used by various programming languages. Lucene provides a very easy and dynamic ways for writing queries to search any text in an indexed document or database.

Lucene API has a strong base core and functionality which remains the same irrespective of the format of the file to be indexed. Lucene is popularly used for single site local search operations or full text search in Internet search engines. Lucene engine is popularly used to handle the twitter search functionality. Twitter search engine handles billions of

queries from different users per day using variations in lucene indexes (22). It uses lucene indexes for appending millions of tweets per day in real time.

Lucene is currently very highly rated full text search engine with huge number of applications built on its base framework. It has been widely used in day today user applications such as iTunes, Outlook search plugins, Comcast, IBM OmniFind Personal email Search, JIRA, LinkedIn, AOL, Eclipse and many more (5). Many applications including Twitter, LinkedIn etc have modified lucene to support their respective functionality and enhanced it to be adaptable for real time search. Lucene is even used as a base to build most of the standalone servers such as Apache Solr and ElasticSearch etc. It supports more then 100 real time applications at a given time (5).

The major process of any information retrieval search engine is Indexing. Lucene framework implements inverted indexing architecture to perform this operation. It basically consists of 5 phases namely : index, segment, document, field and term (10). Each document needs to have unique id to be indexed. The main goal of indexing process is to create index structure for enabling rapid retrieval of documents to a given query operation(search functionality). Any data that can be transformed to text format can be indexed and searched by lucene (10). Lucene can perform indexing on many format of the text files such as word documents, PDF documents, HTML pages and even database tables too. Lucene has implemented partition index, speed index, merge optimization index along with inverted index structure. It also performs incremental indexing as fast as the batch indexing. The index created can contain heterogeneous set of documents. It provides index mappings from terms to documents. Scoring the search results with relevant documents becomes very easy with this structure of indexing.

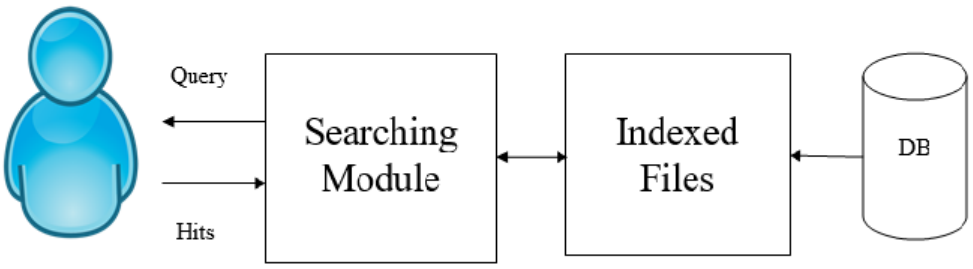

FIG. 1.1. Overview of Full Text Search

The next core operation in FTS is searching the user inputed query. The searching operation is performed on the inverted files generated after indexing is completed. The flow is almost similar to the indexing process but uses different framework classes. The general classes lucene provides for searching are IndexSearcher, TermQuery and TopDocs. It returns a set of documents generated using the relevancy score. Generally, Lucene search API accepts a query converting it to hybrid form of search. The hybrid search query is a combination of vector space search methods, boolean search with relevance ranking weight orientation. Its very important to store the index path and corresponding static properties constants required during system migration. The search index database is built on the base of the query constraints and operations specified. Using only one instance of Lucene IndexSearch will improve the overall speed of the searching.

In this paper, we have implemented the working version of lucene integrated with the eclipse framework. The searching is performed on the database of table present in MySQL of Wamp server. We demonstrate the significant efficiency in searching the free text in a specific column of the table containing large unstructured data. We review the different

methodologies for the same structural implementation in cloud based systems. We analyze the performance results of this implementation with time and accuracy as parameters.

The rest of the paper, we discuss the background and related work of lucene with variations in section 2. Section 3 explains our methodology for implementation of lucene, its methods and functional APIs used for the work. It gives an overview of joint mechanism with the structured framework. It also gives an overview of cloud implementation of lucene framework in recent researches. Section 4 summarizes our results and comparison with several benchmark parameters. We conclude in Section 5 along with improvements and future works in the fusion implementation of lucene and structural databases.

Chapter 2

RELATED WORK AND BACKGROUND

In this section we discuss the background and the related work in the field of Full Text Search, Relational Database, Lucene, Variations in applications of Lucene and Cloud adaptability

2.1 Background

2.1.1 Full Text Search

With the world wide web and commercial large scale databases containing huge amounts of metadata, querying for appropriate texts can be improved with the use Full Text Search. FTS is a frequent technique used by most search engines and web pages today whose fundamental concept is to search each page and index it. In case of any matches or similarities the texts are displayed using the indexing. This method also allows you to view options before actually entering the full content as part of the text entered by the user is queried against the full text (1).

The basic methodology of implementing a FTS in a database is to create one or multiple columns with different characters that may have different datatypes in a table. Each column is indexed using a full-text index and based on the queries entered performs a linguistic search. Once a database is created with full text index users can run full text queries

on the text in columns which helps in identifying the following (18):

- One or more specific words or phrases (simple term)
- A word or a phrase where the words begin with specified text (prefix term)
- Inflectional forms of a specific word (generation term)
- word or phrase close to another word or phrase (proximity term)
- Synonymous forms of a specific word (thesaurus)
- Words or phrases using weighted values (weighted term)

The expanding usage of FTS methodology in search engines is due to its various advantages over traditional database engines. The granular nature of the index facilitates rapid access to specific words and phrases thus improving efficiency. The accuracy of the results obtained which is defined as the relevancy weight is much higher than other traditional querying. Full Text engines using FTS are highly optimized to manage textual data such as proper names, cities, country names. This type of proper names are subject to multiple spelling errors and un-normalized forms which can we well handled by FTS (15). Some of the limitations of using FTS are that when queried for matching results case sensitivity, synonyms and variant spellings are not considered. Another drawback is that languages that don't have word delimiters called ideographic languages such as Chinese, Japanese etc FTS cannot parse them. Examples of applications using FTS are Solr, Google Search Appliance, Sphinx, Lupy, ZEND Framework, Domino, Pylucene etc.

2.1.2 Applications of Full Text Search

Full Text Search is mainly used to for fast retrieval of data from multitude of information in cloud, desktop, databases and other applications. There are many search engines

that use FTS as their basic concept of information search mechanism. One such example is Clusterpoint server which uses the fast performance of FTS to deliver quick response time that do not depend on database structure or size. Database development tools such as SciMoreDB, DBSight and JODA also uses the concept of FTS to find rows in their database.

2.1.3 Limitations of Traditional Database

With the growing amount of unstructured data in electronic format traditional database such as RDBMS became less used. Traditional databases though provide excellent storage and manipulation of structured data i.e data of specific type with less or no redundancy uses more than one tables to store values and returns results as rows of fielded data. One of the major limitation of traditional database is the complexity of the table and storage structure of data in the database. Multiple document tables are created and similar data types are grouped together by storing under same row or record with the components placed in appropriate columns. Well formatted documents and image value types are placed in binary long objects that has special purposes but the number of commands that can be used to retrieve these values are limited (15).

Another major disadvantage of using traditional database structure in search engines is the less flexibility of relational queries. When a relational database is queried to fetch matching textual patterns it returns results in an arbitrary order or sorted based on the field mentioned in the query. Rows that do not match the text in the query are ignored and not fetched in the results. The Full text Search adds an additional component to searching textual content called weight which allow the result set to be sorted based on relevancy score allowing the output to be organized and ordered list of matching records (15).

2.1.4 Lucene

With more development of online web applications the usage of diverse unstructured massive data as content has became common. It has became a task to solve and find the accurate information and content required by people. Lucene supported by Apace Software Foundation is a high performance, scalable and open source information retrieval tool. It functions as a software library that allows data access and management through user friendly API for efficient Information Retrieval (IR). It alleviates the complexity of index and search functionality when used on any web application or software document. Lucene was initially written in Java but now it has been created in various other languages such as PHP, Ruby, Python, C ++, Perl etc (10). The fast retrieval capability of Lucene is due to the concept of indexing wherein a specific keyword in this aspect an index is searched as opposed to searching the entire text. The type of indexing used by Lucene is called Inverted Index as it is keyword centric and inverts a page centric structure. Many documents are created from raw data and used by index for search application to easily understand and interpret. The document model of Lucene helps the search query to check the documents for relevant information. The popularity of Lucene based search engines in comparison with other commercial search engines is due to high efficiency at a much economic scheme. It also supports analyzing and parsing of user entered complex textual expressions while performing simultaneous implementation of searching and indexing.

2.1.5 Applications of Lucene

The main usage of Apache Lucene is in commercial and open source applications where huge amount of unstructured, free format textual content is used. It can be used to incorporate search functionality to any application or software such as website search engine, medical database search engine, email list etc. Today many websites such as

LinkedIn, Wikipedia, TheServerSide uses Lucene as search library for users to retrieve relevant information. The extensive support of powerful queries such as the BooleanQuery, WildCardQuery, RangeQuery that can be used with the tool promotes Lucene to be cutting edge search engine methodology. Many open source web search engine such as Nutch, Eclipse IDE and companies such as Netflix, IBM, Hewlett-Packardm AOL uses Lucene. The various algorithms used for the search operation is extremely accurate and powerful which makes it a suitable tool to be used for many elearning, medical websites and document management system (28). Some of the websites using Lucene as a search tool are ActiveMath, Affidata, Ad Dynamo, Comcast, etc. In recent years many search operations have been built on Lucene such as hit highlighting, auto suggest, faceted navigation, geospatial search etc..

2.1.6 Wamp Server and MySQL

WAMP Server is a web application development environment that has Apache Web server, MySQL database server, PHP, phpMyAdmin installed in it. It is Windows OS based system that automatically configures and installs all files that are needed to build web applications intuitively. WAMP is also used to perform internal testing wherin the developer can test webpages without deploying them live through the Apache HTTP Server (30). WAMP can import data from both CSV and SQL and can export data to various formats such as CSV, SQL, PDF, XML, ISO/IE C 26300 Open Document Text and Spreadsheet (8).

MySQL is a high speed relational database management system that handles exceptions, errors, transaction and is used as a building block to create the backend of web applications. WAMP uses the MySQL tool installed in it to handle the database of the dynamic website and supports most of the MySQL features such as management of stored procedures, triggers, users and privileges. It also supports database, fields, tables and index modification and maintains them with proposals on configuration server. With WAMP the

database used for the website can be tested efficiently as complex queries can be created in the server using Query-by-example (QBE) to search records globally. The data stored using MySQL can be transformed by WAMP using predefined operations that can even show a BLOB data as a download link on the front page. Multiple servers can be administered using WAMP and with phpMyAdmin handle the SQL server to perform create,edit, modify functions (8). The main advantage of using a WAMP server to build and test websites is due to the easy of usage where everything is available with one click away and no additional configuration is needed.

2.1.7 Support of Full Text Search in MySQL

Text can be searched in MySQL using operators such as LIKE and REGULAR which has some drawbacks and performance limitations when the the columns and rows are increased and have large texts. To mitigate the above limitation and improve the performance of information retrieval MySQL uses Full Text functionality to query full texts against character based data. MySQL contains clustered and non clustered indexes to query most columns in a table but these indexes cannot query large object column data types. A Full Text index supported by FTS allows columns configured with char, varchar, varbinary, image to be effectively queried against full-text user requests. Some of the important features of MySQL full-text search are (2):

- Native SQL-like interface: SQL-like statement are used for the full-text search.
- Fully dynamic index: MySQL automatically updates the index of text column whenever the data of that column changes.
- Moderate index size : Memory storage used for the index is not very large.
- Fastest means of method to search complex queries.

There are different types of Full Text Searches that are supported by MySQL :- a. Natural Language FTS b. Boolean FTS c. Query expansion Search. The Natural Language FTS understands the text in the query as a human language such as a free text and no special operators are needed. MATCH function is used to find the string in the table with multiple text collection. The Boolean FTS interprets the string used in search in complex query language. The string will have operators to understand the relation of the complex strings whether they need to be present in the search rows (4).

Major advantage of using FTS in MySQL is reliability and flexibility of retrieving accurate results. Since natural language search is facilitated by FTS it is much more user friendly. The inverted index allows improved performance and scalability than normal clustered indexes. Some limitations of FTS are that it might be performance heavy to insert large datasets in a table with full text index and that it only can be used on MyISAM and InnoDB tables and are not suitable for partitioned tables. Also index hints are limited in Full Text Search as compared to normal searches. There are few default words called stop words that are ignored which might be a problem if they are queried.

2.1.8 Indexing algorithms for search engines

The main purpose of a search engine is to search the front end of application using indexes in the back end of the document. The column indexes parses through each row of the table and find appropriate results using different algorithms. The different algorithms used by search engine index are as below:

- Word Count algorithm
- Unique word algorithm
- Comments (Manual entry) Tracing algorithm

- Bold Text representation algorithm
- Italics Text representation algorithm
- Links algorithm
- Heading, sub heading algorithm

Word Count Algorithm counts the total maximum number of times a word occurs on the file and the value is recorded. This is useful in calculating the frequency of a term in a document. The unique word algorithm compares files to find whether the words are present in one or multiple files. The words that are present in fewer files are called unique words and they calculate the inverse document frequency. Comments Algorithm searches for comments provided by the author that describes the content of the document accurately. Bold text Algorithm indexes file based on the words written in bold. The link algorithm links files based on hypertext and such that one file having hypertext relation with one has the same relation with another if those are related (23).

2.1.9 FTS support in Cloud Storage

With the continuous growth of information flowing across the world wide web online data storage called cloud storage has became an effective storage option. The traditional database was not able to handle the scalability, search in real time and efficient searching results due to increase in quick growing and unstructured information. Elastic search engine which is built on top of lucene is designed to perform search operations on cloud platform to solve these issues (29). Big data tools like Hadoop can be combined with open source engines to search data efficiently in cloud storage. The index is refreshed in every one second to obtain the real time search in case of elastic search engine. For the searching

operation to work correctly the index file needs to be downloaded, updated and reload it back to the cloud storage.

One of the most important factors that needs to be considered while providing Full Text Search in cloud storage is security. It is very important to support FTS without providing un-encrypted data to the server side cloud. One way to achieve security is to download the encrypted data to client side, decrypt it and perform the searching operations. Most of the FTS techniques in cloud depends on secure index. The efficient way is to create index for all the information stored on the cloud and stored it in client side. This approach results in faster searching time since we search through the index instead of entire data in the cloud. For every search request the client side will check through the local index and retrieves the file directly if it is present locally or else downloads it from the cloud storage (3).

2.2 Related Work

In this paper we discuss various implementation of Lucene and researches done on the performance of lucene with fusion of other techniques. There have been numerous research on the review of working architecture of Lucene (17). They have performed various literature survey of different implementation of Lucene along with providing brief class architecture of Lucene Framework. They summarize that Lucene is one of the best search and retrieval open source search engine with vast adaptations in real time (17). They submit the strength, weakness and findings of different lucene applications. The experiment analysis proved that searching time for Desktop full text search based on lucene in 550000 words document size was 1798 ms (17).

We have given a brief introduction about FTS and its applications in the above

sections. Many papers have been published on FTS in recent time due to its importance in the current information gathering. Many algorithms and variations in indexing have been performed with FTS to improve the speed and efficiency of information retrieval. Full Text search has been combined with access control using generalized suffix tree to obtain less memory on indexing and rapid query results (32). The combination of access control with FTS acts as filter for obtaining relevant documents. Hence it will reduce the number of search results depending on user rights (32). This is widely used in social media to control the search results depending on access right of each individual.

It has been analyzed that Lucene Information retrieval is much more faster than traditional string retrieval technique. The experiment results and analysis conducted by Gao, Li and Dong (12) proves the above claim about speed of retrieval results. The amount of time taken by Lucene for a document with 250,000 words is 75 ms as compared to string retrieval consuming 1988 ms (12). The retrieval time increased linearly with the size of the original document. An individual intranet information retrieval service has been implemented by Hongyin and Xuelei(13) using Lucene eclipse plugin. They used web crawler and different parsers to convert the gathered information into plain text data. The experiment results shows that it has very good retrieval efficiency and performance with a better indexing structure (13).Similarly, Lucene has been used to retrieve office documents with heterogeneous data source environment.

Sun Lincheng(et.al 2011) implemented a dot net version of Lucene to collect educational data information from the digitized versions of new paper. Many standard web scraping and crawler extensions were used to convert the information obtained to a free text data structure. The compared and analyzed the standard fuzzy search against relational database and fuzzy search against full text search engine developed using Lucene.

The experiment results and performance analysis showed that for about 550,000 records the response time for query operation was 200 ms using DotLucene. The response time of traditional database methods for 550,000 records was 750 ms (20). The experiments also proved that with increase in the size of records the amount of response time in traditional database was growing very fast compared to DotLucene implementation (20).

Many desktop search engines have been implemented using Lucene to find personal documents in a individual PC (9). They implemented and evaluated Desktop full text search system using Lucene which provided unified access to the documents in personal computer depending on their contents (9). They indexed every document of different formats into a simple text data and succeed in searching the document location based on their contents. Zhang and Lin-Li(et al. 2009) implemented Lucene search engine with new search algorithm instead of using the standard Lucene search API package. They combined Vector Space Model(VSM) and Page Rank algorithms to obtain a new efficient retrieval sorting algorithm (33). The gave a brief overview of merging the source documents to generate efficient and rich indexing performance. The experiment calculated the Average Satisfaction Degree(ASD) of different users for all the three searching algorithms. The experiment was performed on document size of 46855 pages. The experimental analysis proved that the ASD for the fusioned sorting algorithm(65.7) is better than Page Ranking(46.2) and Lucene(37.9) sorting algorithms respectively (33).

Paper duplicate detection system in a magazine was implemented by Ding, Yi, Xiang(et al. 2010) to detect the duplications in title, content of the magazine with size of 2000 pages. The design was based on lucene architecture for performing information indexing and data search. They also implemented keyword highlighter to improve the performance of indexing and searching. They used chinese analyzer to support full text search in order

to increase the accuracy of search in chinese language. The experiments showed the parsed 250,000 paper magazine to search for a particular chinese keyword yielding in 148 results of different documents for the keyword consuming 300 milliseconds (11). The analysis showed that Lucene search implementation was yielding faster search results compared to database full text search.

Qian, Wang (et al. 2010) implemented free text search based on both Apache Lucene and Oracle text. They experimented the implementation against 700 million tuples and the analysis proved that lucene was better for huge amount of data. Lucene was twice faster than Oracle text in retrieving the query results (25). The analysis summarized that disk I/O plays a critical role in keywords search and optimal use of I/O bandwidth can improve the performance rapidly (25). Shi and Wang(et al. 2014) combined both Lucene and Oracle database to implement free text searching. The results showed that recall ratio and precision of query results can be improved drastically using this combination (27). The proposed combination algorithm exhibited improved efficiency in Indexing the database and faster search results. They also implemented specialized indexing for Date and Time format records (27). The storage of data also plays a very important role in full text searching. The multiple experiments conducted showed that simple file system is more efficient if the data size is less than 256 kilobytes over the database storage structure (26). The experiments and analysis was performed using NTFS file system and SQL server. Many researchers performed full text information retrieval using Lucene, MySQL Server and PostgreSQL. Lucene was integrated to perform full text search on Turkish text documents and compared the results against full text capabilities of relational database systems (7). The experiments were conducted on Milliyet collection which consists of 408305 documents, 72 topics of 800MB in size. The best average indexing time and optimal performance was provided by lucene compared to other two implementations (7).

Currently, many research has been undertaken to implement and measure the performance of open source search engines in cloud storage. It has been implemented on major standard cloud platforms such as Amazon EC2, Microsoft Azure and many more. Khaled Nagi (et al. 2015) implemented a benchmarking system using different search engines to test the indexing and searching functionality over 49 GB of textual content (24). The experiment and analysis was performed on 4 different search engines namely: SolrCloud, SolrCloud on Hadoop, SolrCloud on Cassandra, Lucene on MongoDB. The analysis reports that the throughput of Cassandra for indexing is the best compared to other 3 systems and Mongo DB based system had the slowest indexing system (24). The results proves that Solr gives superior indexing and fast searching operations compared to search engines built on NoSQL databases. Will, Ko, Witten(et al. 2015) proposed a system which supports full text retrieval in cloud storage (3). They applied boolean and ranking operations to the entire list of documents in cloud storage by treating the multi term queries as separate index retrievals. They also encorporated to perform approximate search to avoid spelling errors in searching.

Chapter 3

METHODOLOGY

3.1 System Architecture

The overall system architecture revolves around Lucene engine and Relational database tables. Full text search is provided to all type of documents which can be converted and stored as relational database tables. The main modules of any search engines is Index creation and efficient searching mechanism. The full functional block diagram of the retrieval architecture is as defined in Figure 3.1.

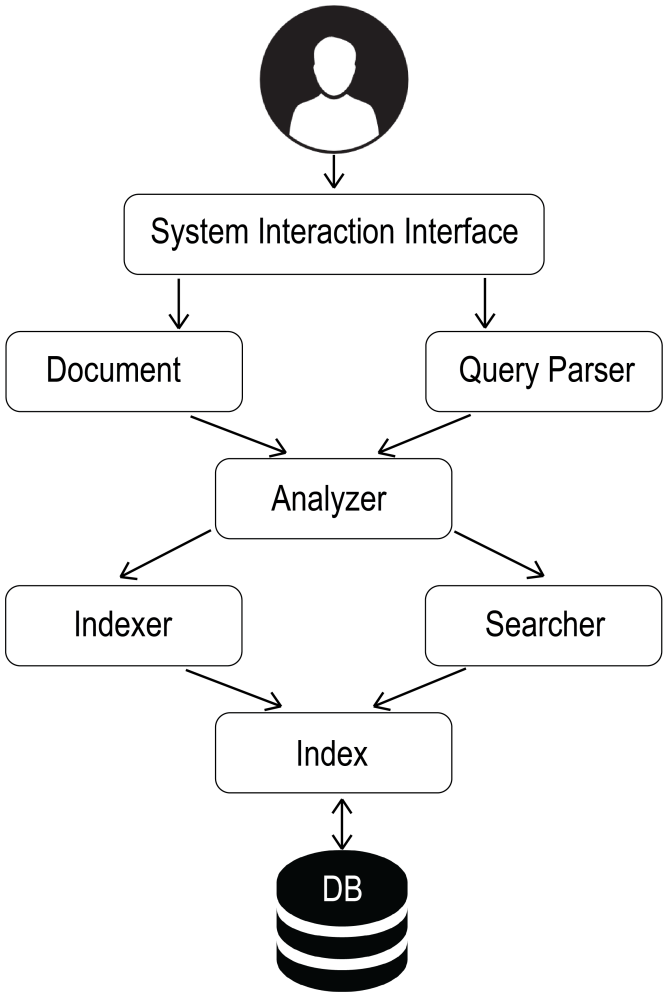

FIG. 3.1. Functional Block Diagram of Lucene

The entire Lucene Framework is composed of 7 different modules as described in Figure 3.2. In order to implement the full text searching functionality in relational database Lucene provides simple functional calling interfaces for information access and management. Indexing and searching are the two primary functions in any search engine. Lucene contains the entire index engine and query engine within itself to support searching in different applications. Lucene is widely used not only because of its simple and efficient indexing functions but also due to its ability to be integrated into many software systems

and web applications.

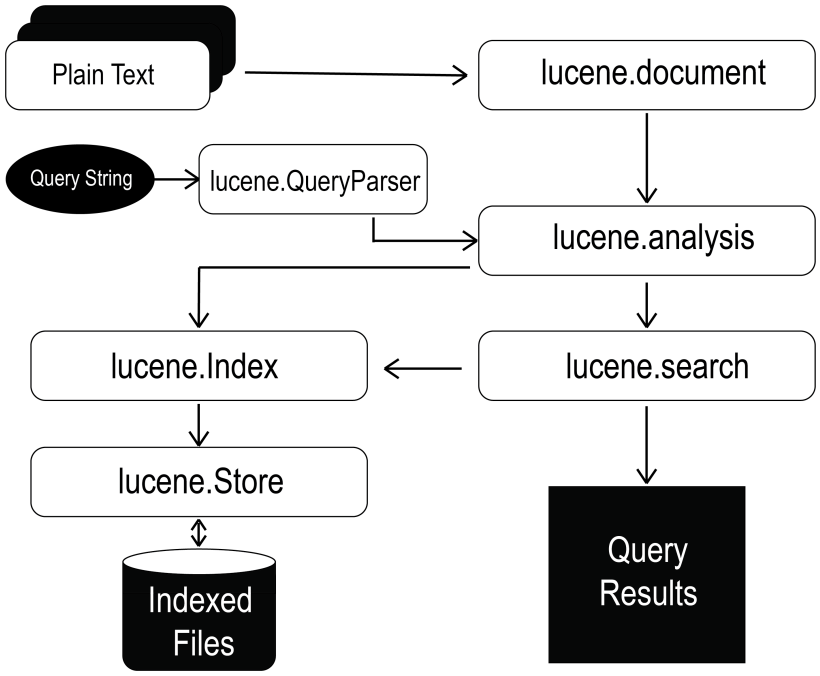

FIG. 3.2. System Architecture of Lucene (33)

The functionalities of each package is described in Table 3.1.

Package Name	**Functionality**
org.apache.lucene.document	Fields and Document structure management
org.apache.lucene.analysis	Segmentation and Stop word removal
org.apache.lucene.index	Index creation and management
org.apache.lucene.queryParser	Query parsing and analysis
org.apache.lucene.search	Structure representation for Queries
org.apache.lucene.store	Binary I/O API for storing data
org.apache.lucene.util	Utility classes

Table 3.1. Core Packages of Lucene (11)

- Package org.apache.lucene.document : This module is responsible for management of indexable fields. The fields are further divided in text and data fields. A document is just collection of fields where fields represent the logical data to be indexed.

- Package org.apache.lucene.analysis : This module is responsible for language processing and lexical analysis. It performs different word segmentations for the actual text contents. It also supports stop words removal for the indexed contents.

- Package org.apache.lucene.index : This module is responsible for creating, updating and deleting an index. It generates index for every word in the text document. It relates the data pointed by the index.

- Package org.apache.lucene.queryParser : This module is responsible for parsing a user input query and transforming it to a string object. It provides functionalities to implement operational keywords and logical negations.

- Package org.apache.lucene.search : This module is responsible for the retrieval of matching documents based on the user input query. It simply collects index results based on the query parameters passed to it.

- Package org.apache.lucene.store : This module is responsible for providing the underlying I/O structure. It provides classes for storing persistent data, file system directory and also in-built memory resident data structures.

- Package org.apache.lucene.util : This module is responsible for providing helpful extra data structures and utility classes. The utility classes supported contains PriorityQueue and FixedBitSet functionalities.

3.2 Indexing Mechanism

Indexing is the core functionality in Lucene search engine. Lucene implements inverted indexing approach. Inverted indexing points words to documents whereas sequential index relates documents to words. Inverted index improves the efficiency of information retrieval since it outputs all the related documents if keyword is known.

The structure of lucene index is basically divided into 5 stages levels : Index, Segment, Document, Field, Term (10). Figure 3.3 gives the categorization of Lucene Index structure. The smallest unit term consists of string and information such as frequency in each document. During analysis process, Lucene performs many operations on Terms such as normalizing, stemming, removing common words, lemmatization etc. This analysis and process is termed as Tokenization. The small bits of data pulled from the input text is called as tokens. Tokens combined with fields names compromise Terms.

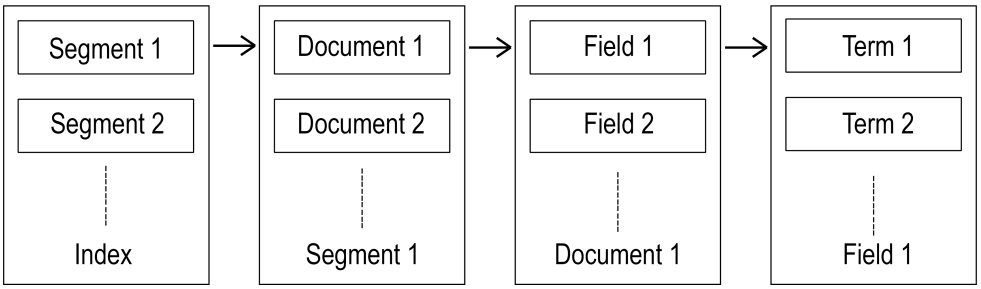

FIG. 3.3. Index Structure of Lucene

To index any free text, the data needs to be converted into Documents and its corresponding fields . We use IndexWriter to analyse and accept those Document objects. The data is stored in Inverted index structure after calling IndexWriter functions. We can optimize the index creation by setting the merge factor parameters. One of the performance

bottleneck of indexing is it writes the indexes into disk which can be handled using merge-factor. Lucene also provides APIs for updating and deletion of indexes.

3.3 Searching Mechanism

The searching operation in Lucene is performed depending on the user query keywords. It is exactly same as other commercial engines such as Google but uses different retrieval mechanism. Initially Lucene search accepts the query string then it sends the query to QueryParser. QueryParser parses the input and simplifies it by performing language processing to extract operation keywords. Later it performs searching in the indexed database and returns the matching documents for the input keyword. Figure 5 explains the mechanism of searching in Lucene.

Lucene provides functions to optimize searching. We can limit the number of related documents by using TopDocs. It can retrieve only 10 top most matching documents in 1000 of matching documents by specifying the parameter. We can also highlight the number of hits and specify related contents that need to be displayed. For example in one of our experiments we are displaying Customer ID along with the document contents.

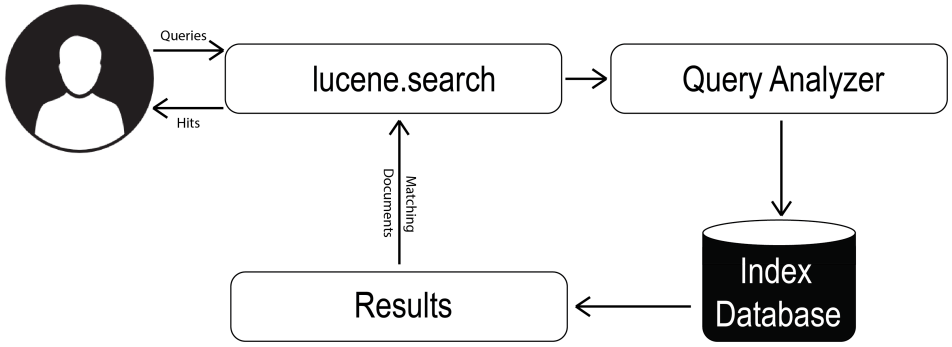

FIG. 3.4. Lucene Searching Mechanism

3.4 Cloud Adaptability

Apache Solr is an search engine built on top of the Lucene core engine. One of the main features of Solr compared to Lucene is it provides REST APIs for JSON format of data. SolrCloud was released in 2012 as an application to Solr search engine (29). It supports sharding and replication features. Solr provides the user to configure different shards in their console connected to the cloud. Solr provides functionality to convert its documents to Lucene documents.

Solr typically uses Cassandra for storing the indexes instead of file system. The user can use CassandraDirectory and its associated classes for index storage. It provides SolandraIndexReaderFactory to retrieve the contents form the indexed documents stored in Cassandra nodes. The information is exchanged between the Cassandra clusters every second. The cluster can have multiple nodes and any node which accepts the user request acts as coordinator. NetworkTopologyStrategy class specifies the replication factor in each cluster. The coordinator forwards the request to the node which contains the related information about the request. Cassandra is known for its fault tolerance, scalability and decentralized features (3).

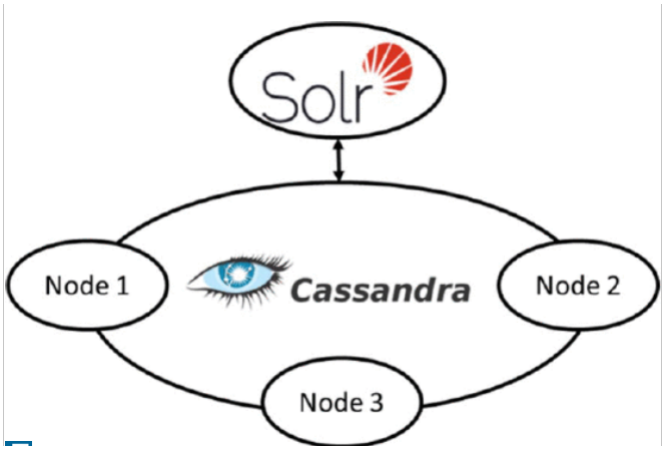

FIG. 3.5. Solr- Cassandra Implementation Structure (29)

SolrCloud integrated with ZooKeeper provides index replication, handling distributed queries, fault tolerance, load balancing and other specialized features. SolrCloud is a combination of multiple Solr nodes with a collection of shards. ZooKeeper offers easy maintenance of data and efficient handling of index distribution. It distributes all the indexed contents and also the query requests across all the nodes providing perfect load balance. The number of replicas in each shard determines the strength of fault tolerance. The physical cores in clusters represents the logical shards in SolrCloud. The user can also specify which shard the documents need to be stored (6).

Microsoft Azure uses Lucene to perform full text search. It performs query parsing, lexical analysis, document matching and scoring to retrieve efficient results. It uses the similar architecture as Lucene to perform the free text search. It provides REST API to perform all the above operations. Query parsing categorises the input query into term query, phrase query and prefix query. The user has the option to set the query parser to simple and full. Full query parser handles wildcard, fuzzy, regex and full scoped queries (19).

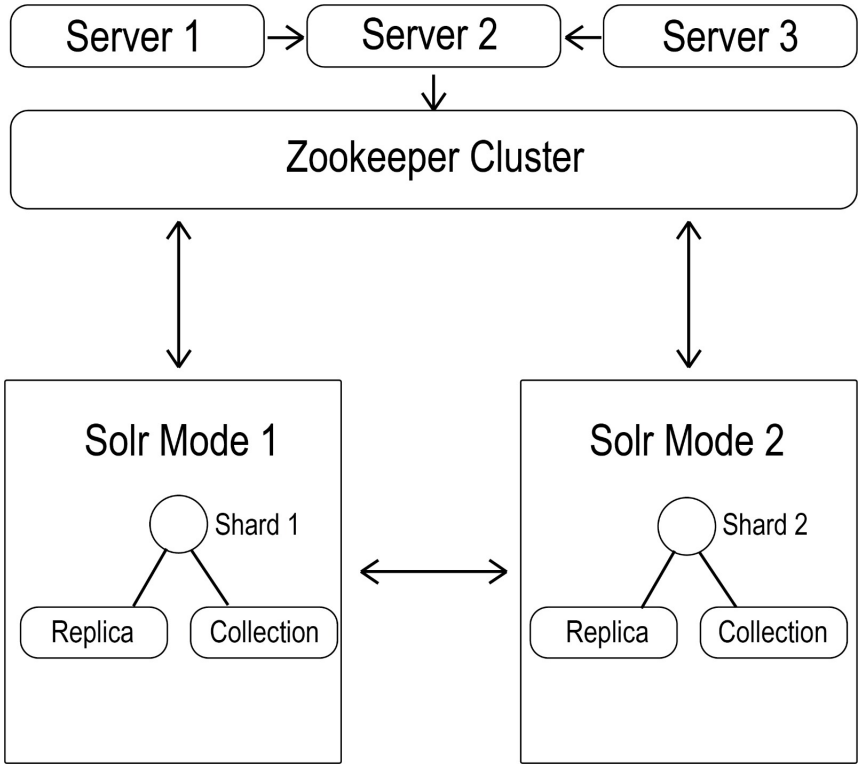

FIG. 3.6. SolrCloud Implementation Structure

Chapter 4

RESULTS

This section provides different experiments and performance analysis of Lucene search engine on huge customer complaint dataset of size 1 million rows. The main goal was to provide full text search capability for Customer Complaint database using Lucene architecture. After successfully implementing lucene we also compare the performance of Lucene and MySQL full text indexing. The section also shows different types of queries supported by lucene and their performance respectively.

4.1 Case Study and Performance metric

The case study of customer complaint database has a huge table of 1 million rows. It consists of 12 columns explaining the complaint in detail. The goal is to provide Full text searching for Issue, Sub Issue and Complain Narration columns. We were able to retrieve all the details of a complaint depending on a keyword in one of these columns Ex: Debit in Issue Column. We have created separate field indexes for each column.

The experiments conducted focuses on lucene efficiency in performing full text search on Customer complaint relational database. The performance metrics considered here are Indexing time and Searching time. We also compare both of them against MySQL full text

searching capability. We also analyze the performance with increase in the size of dataset.

4.2 Experimental Environment

The experiments was performed on dedicated CPU with windows 10 operating system, i7 with 2.6 GHz processor, 12 GB RAM with an hard disk of 1TB. We used open source Apache Lucene 7.3.1, MySQL community server 5.7.21. We used the latest version of all the systems for conducting our experiments to support latest features. The development platform used was Eclipse Java Photon with MySQL java connector 8.0. All the programs were implemented in Java programming language.

4.3 Lucene Performance Evaluation

After deploying lucene search engine on eclipse we have conducted different tests for analysing indexing and searching performance. The Figure 4.1 shows the output of our program when we enter loan for searching in indexed database. It took 4.76 seconds for indexing the test database table with 100,000 rows. The searching time taken to find 4582 hits was 34 ms.

```
Indexing to directory 'MMapDirectory@C:\testlucene
Time taken for indexing : 4766 ms
Time taken for searching : 34 ms
Found 4582 hits for Issue: loan
Number  Complain ID     Contents
-------------------------------------------------
1.      42864           Repaying your loan
2.      104285          Repaying your loan
3.      42864           Repaying your loan
4.      104285          Repaying your loan
5.      42864           Repaying your loan
6.      104285          Repaying your loan
7.      137137          Repaying your loan
8.      361206          Getting a loan
9.      2364581         Getting a loan
10.     296040          Repaying your loan
11.     418456          Repaying your loan
12.     469745          Repaying your loan
13.     202925          Repaying your loan
14.     105255          Repaying your loan
15.     335057          Repaying your loan
16.     2103173         Getting a loan
17.     196874          Repaying your loan
18.     42498           Repaying your loan
```

FIG. 4.1. Lucene Search output for keyword search

We performed various keyword searches for the above test set to obtain different hit ratios. Table 4.1 clearly explains the relationship between hit ratio and search time consumed.

Query String	**Number of rows containing the query string**	**Total searching time (ms)**
credit	15298	43
owed	7781	37
loan	4582	30
score	1314	25
company	795	20
fee	38	15

Table 4.1. System Performance with different keyword queries

We have analyzed the performance time of lucene indexing with gradual increase in the size of data set. Table 4.2 and Figure 4.2 shows the comparison results of the indexing times with increase in number of rows. We also tested the Lucene performance on Index searching with increase in the original data set size. In figure 4.2 we observe that indexing time gradually increases with increase in the data set size. The Index searching performed in Figure 4.3 is done on loan keyword search in issue column of the dataset. The experiments were performed until the database contained 1 million rows.

The below figures and table shows the Indexing performance of Lucene :

Number of Rows	Indexing Time of Lucene (ms)
100,000	4384
200,000	7037
400,000	13753
600,000	20890
800,000	26145
1,000,000	38133

Table 4.2. Indexing Performance of Lucene with increase in dataset size

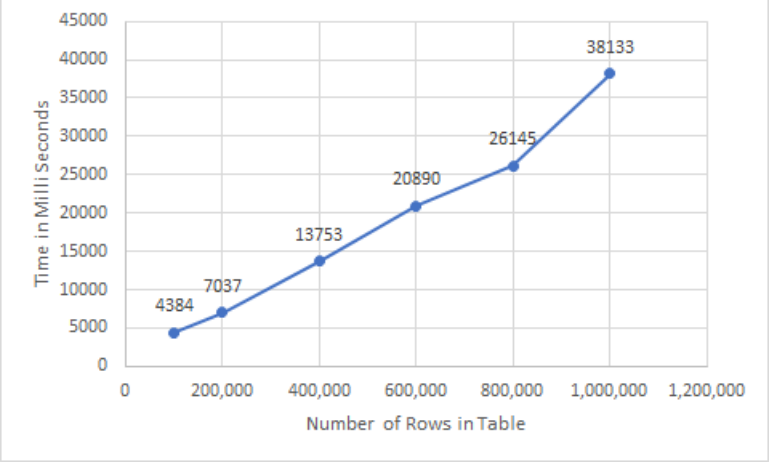

FIG. 4.2. Lucene Indexing Performance

The below figures and table shows the searching performance of Lucene for loan keyword in the generated index files :

Number of Rows	Number of Hits	Searching time in Lucene (ms)
100,000	4582	31
200,000	8911	47
400,000	17557	57
600,000	28144	63
800,000	36804	70
1,000,000	44138	77

Table 4.3. Searching Performance of Lucene with increase in dataset size

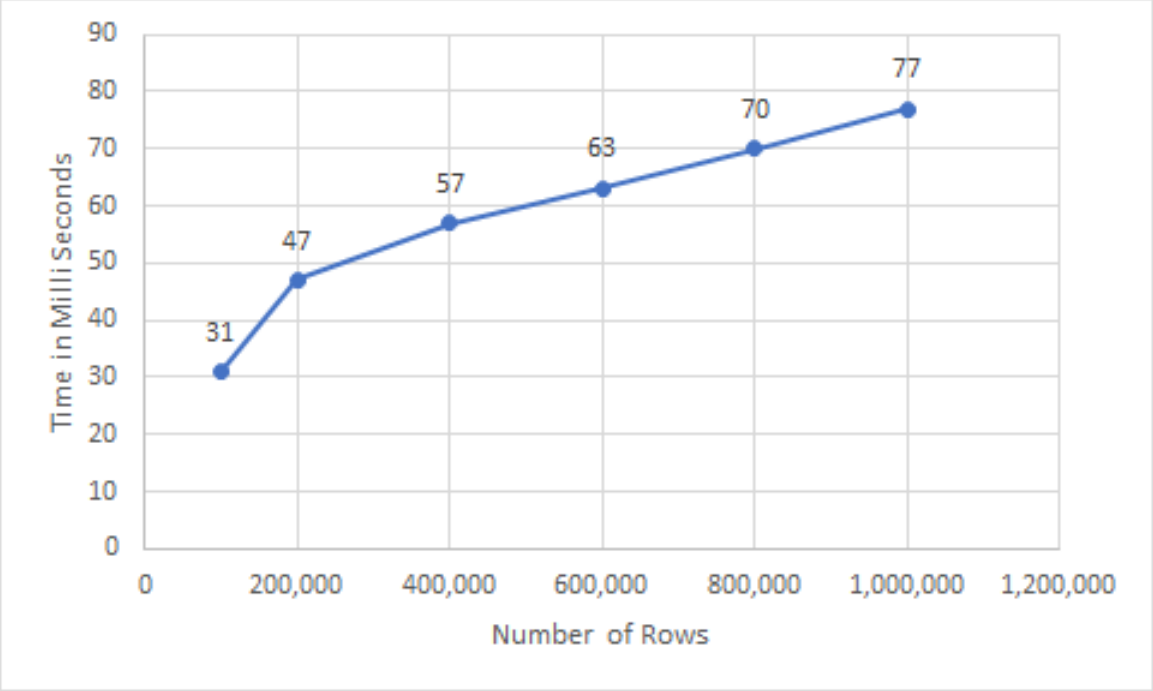

FIG. 4.3. Lucene Searching Performance

We have also implemented Full Text indexing in MYSQL. Below table and graphs shows the comparison between Lucene and MySQL indexing performance with in crease in data set size :

Number of Rows	Indexing Time of Lucene (ms)	Indexing Time in MySQL (ms)
100,000	4384	12371.8
200,000	7037	23289.6
400,000	13753	51147.7
600,000	20890	80735.9
800,000	26145	100817.1
1,000,000	38133	145069.8

Table 4.4. Comparison of Lucene and MySQL Indexing Performance

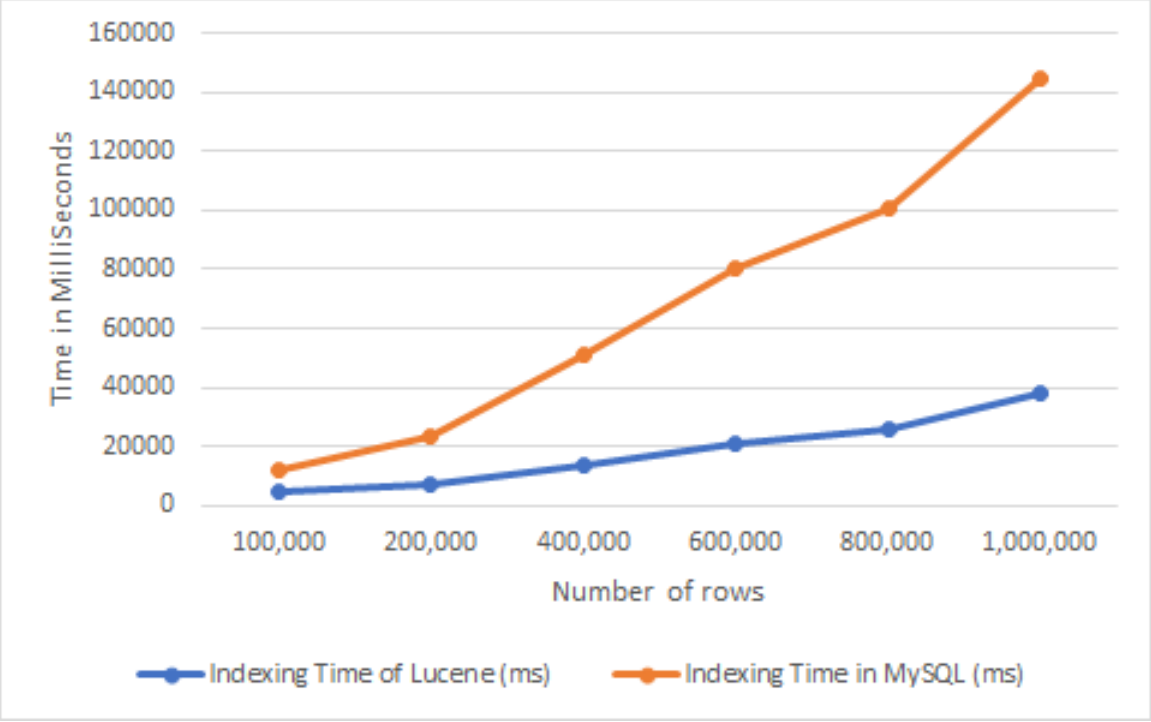

FIG. 4.4. Lucene Indexing vs MySQL Indexing

We have performed the exact keyword searching in both lucene and MySQL for a dataset with 200,000 rows. Table 4.5 shows the search performance comparison between Lucene Full text searching and MySQL Full Text searching :

Query String	Search Time in Lucene (ms)	Search Time in MySQL FullText Index (ms)
credit	95	400
owed	63	120
loan	62	380
score	34	040
company	59	110
fee	46	No matches

Table 4.5. Comparision of Lucene and MySQL Index Searching Performance

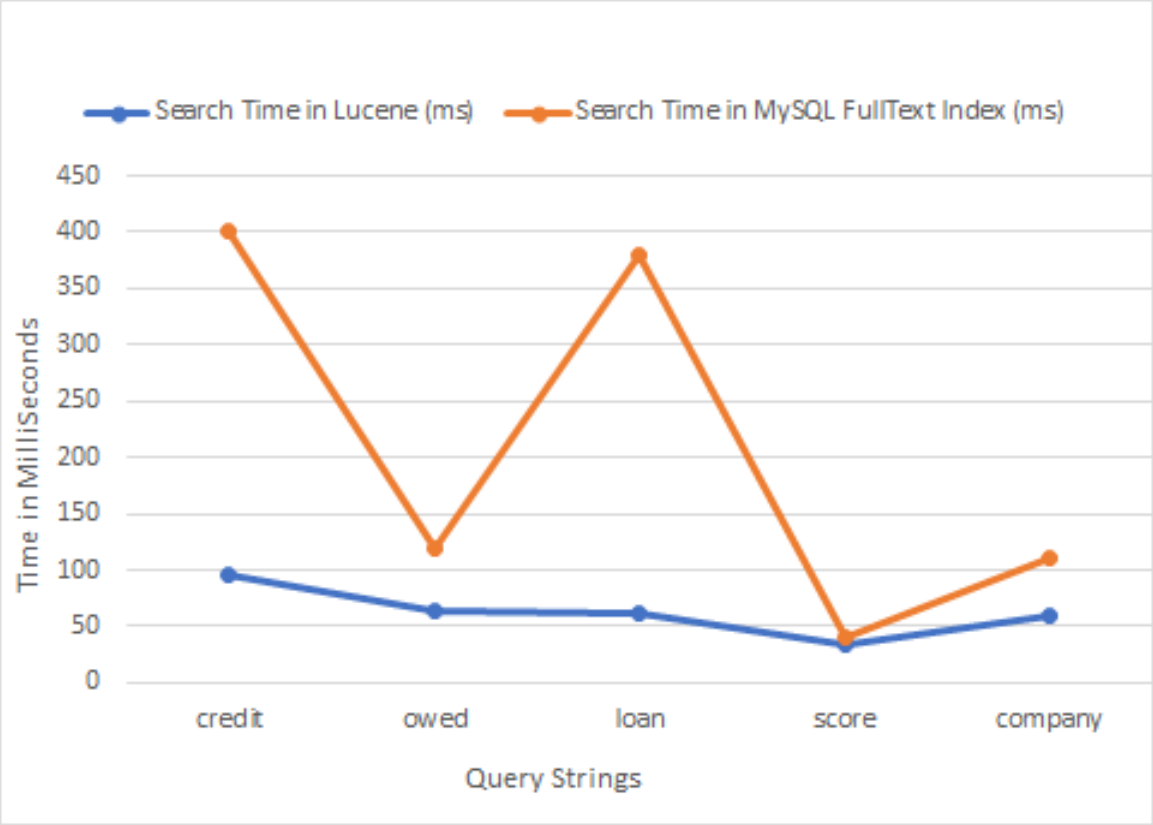

FIG. 4.5. Lucene Searching vs MySQL Searching

Chapter 5

CONCLUSION AND FUTURE WORK

Over the last decade the methodologies in storing the digital data has changed to a huge extent. Currently, most of the applications store both relational structural data and non-structural data at the same time. The requirement to efficiently mine these data for information plays a crucial role in real time scenario. It is very significant to have full text search capabilities even for the relational databases.

Lucene is full-text information tool kit which can be easily integrated into websites and standalone applications to provide efficient searching capabilities. It is highly reliable and has been widely used in numerous industries. The system design of lucene supports the object oriented architecture and vastly know for its ease of integration. In this paper, we explain the system architecture, indexing mechanism and searching mechanism of Lucene in depth. We implement a full text searching application based on lucene on relational database. In this paper, we try to provide full text searching mechanism for Customer complaint database using lucene. We have evaluated the indexing time and searching time of Lucene with increase in the database size. We also analyze the index and hit ratio for keyword specific experiments. The whole experiment was performed on database containing more than 1 million rows. It provides specific information details about a customer

complaint depending on keywords present in the complaint description. The experiments shows very good indexing and retrieval mechanism for a small standalone application.

We also implement full-text searching using MySQL built-in inverted Index support. We analyze the indexing and searching results between MySQL and Lucene implementation. Our experimental results shows that Lucene implementation is efficient then MySQL implementation. We also review current cloud implementations of Lucene and its different applications.

In the future, we would like to further improve the performance of lucene by implementing the indexing of data into a cloud rather then the local disk. We can also improve the searching results by merging the indexes and creating a load balance between them. We need to optimize the index structure for efficient searching results. We also need to support different query search mechanisms in the future implementation. We also need to analyze the memory utilization and cloud implementation in detail.

REFERENCES

[1] Full-text search: Techopedia.

[2] Introduction to mysql full-text search.

[3] Manage massive amounts of data, fast, without losing sleep.

[4] Mysql full text search.

[5] Powered by lucene.

[6] Solrcloud.

[7] Ahmet Arslan and Ozgur Yilmazel. A comparison of relational databases and information retrieval libraries on turkish text retrieval. *2008 International Conference on Natural Language Processing and Knowledge Engineering*, 2008.

[8] Talha Asif. Wamp server tutorials, May 2012.

[9] Mohit Bhansali and Praveen Kumar. Searching and analyzing qualitative data on personal computer. *IOSR Journal of Computer Engineering*, 10(2), 2013.

[10] Xiaomei Chen and Lizhen Xu. An educational resource retrieval mechanism based on lucene and topic index. *2016 13th Web Information Systems and Applications Conference (WISA)*, Sep 2016.

[11] Yuehua Ding, Kui Yi, and Rihua Xiang. Design of paper duplicate detection system based on lucene. *2010 Asia-Pacific Conference on Wearable Computing Systems*, Apr 2010.

[12] Rujia Gao, Danying Li, Wanlong Li, and Yaze Dong. Application of full text search engine based on lucene, Oct 2012.

[13] Yan Hongyin and Qi Xuelei. Design and implementation of intranet search engine system. *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, Aug 2011.

[14] ILya Katov. How to improve database searches with full-text search in mysql 5.6 on ubuntu 16.04, Oct 2017.

[15] Miles Kehoe. Contrasting relational databases and full-text search engines, Jun 2004.

[16] Marc Krellenstein. Full text search engines vs. dbms, Sep 2009.

[17] Sanu Lakhara and Nidhi Mishra. Desktop full-text searching based on lucene: A review. *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, Sep 2017.

[18] douglas Laudenschlager. Full-text search, Apr 2018.

[19] Janusz Lembicz. Full text search engine (lucene) architecture in azure search, Apr 2018.

[20] Sun Lincheng. A large-scale full-text search engine using dotluence. *2011 IEEE 3rd International Conference on Communication Software and Networks*, 2011.

[21] Xiaozhu Liu. Efficient maintenance scheme of inverted index for large-scale full-text retrieval. *2010 2nd International Conference on Future Computer and Communication*, May 2010.

[22] Lucidworks. How twitter uses apache lucene for real-time search, Sep 2015.

[23] D. Minnie and S. Srinivasan. Intelligent search engine algorithms on indexing and searching of text documents using text representation. *2011 International Conference on Recent Trends in Information Systems*, Dec 2011.

[24] Khaled Nagi. Bringing search engines to the cloud using open source components. *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Nov 2015.

[25] Liping Qian and Lidong Wang. An evaluation of lucene for keywords search in large-scale short text storage. *2010 International Conference On Computer Design and Applications*, Jun 2010.

[26] Russell Sears, Catharine Van Ingen, and Jim Gray. To blob or not to blob: Large object storage in a database or a filesystem? 01 2007.

[27] Xiujin Shi and Zhenfeng Wang. An optimized full-text retrieval system based on lucene in oracle database. *2014 Enterprise Systems Conference*, Aug 2014.

[28] Amol Sonawane. Using apache lucene to search text, Aug 2009.

[29] Urvi Thacker, Manjusha Pandey, and Siddharth S. Rautaray. Performance of elastic-search in cloud environment with ngram and non-ngram indexing. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Mar 2016.

[30] Nina Wang. Building the wamp platform, 2011.

[31] Wikipedia. Inverted index, June 2018.

[32] Ahmad Zaky and Rinaldi Munir. Full-text search on data with access control using generalized suffix tree. *2016 International Conference on Data and Software Engineering (ICoDSE)*, Oct 2016.

[33] Yong Zhang and Jian-Lin Li. Research and improvement of search engine based on lucene. *2009 International Conference on Intelligent Human-Machine Systems and Cybernetics*, Aug 2009.