Abstract

Instant search is an emerging information-retrieval paradigm in which a system finds answers to a query instantly while a user types in keywords character-by-character. Fuzzy search further improves user search experiences by finding relevant answers with words similar to query keywords. A main computational challenge in this paradigm is the high speed requirement, i.e., each query needs to be answered within milliseconds to achieve an instant response and a high query throughput. At the same time, we also need good ranking functions that consider the proximity of keywords to compute relevance scores.

In this project, we study how to integrate proximity of information into ranking in instant-fuzzy search. We adapt existing solutions on proximity ranking to instant-fuzzy search. A naive solution is computing all answers, and then ranking them. But this solution cannot meet the high-speed requirement for large data sets when there are too many answers. Therefore, there are studies of early-termination techniques to efficiently compute relevant answers. To overcome the space and time limitations of these solutions, we propose an approach that focuses on common phrases in the data and queries, assuming records with these phrases are ranked higher. We study how to index these phrases and implement an incremental-computation algorithm for efficiently segmenting a query into phrases and computing relevant answers.

Table of Contents

List of Figures

Chapter 1

Introduction

An emerging information access paradigm is instant search. Instant search returns the answers to the users based on a partially typed query. Many users prefer the experience of seeing the result for their searches instantly and create their queries based on the suggestions. According to research by the University of California, Irvine users find this type of information retrieval more helpful and faster, which allows them to get the desired results with lesser effort.

Majority of the users expect to see the search results even before they hit the search button. This can be achieved by instant search method where the search algorithm predicts the partially typed query and returns the results. Accuracy of the answers also plays a very important role in this search paradigm. Fuzzy search makes sure that it returns the appropriate results in case of typing error or if there is more than one correct answer to a search query. The server needs to return good quality and accurate search results in a very limited time. Finding relevant answers is as important as the computational speed.

In this project our main idea is to implement an efficient instant fuzzy search by traversing through the records and stop the exploration once we find the set of approximately accurate answer to the search query. We also implement incremental computation in order to get the result immediately instead of computing the results for each keystroke.

A query will be divided into segments so that these segments can be executed in an order to get the result as quick as possible. Each keystroke is treated as a query; thus high speed computation and retrieval is required for a high-end input query. We will implement and try to test this search paradigm with real data-sets.

For the current implementation we take three simple approaches for searching a query and merge them to create a unified search engine for a given domain/website.

1. Instant Search
2. Fuzzy Search
3. Search with Proximity Ranking.

The project report is organized as follows:

Chapter 2 discusses about the search engines, purpose and scope, existing system. It also defines the instant search, fuzzy search and proximity ranking.

Chapter 3 discusses about the proposed solution, the modules involved, software and hardware requirements and an overview of the programming languages and software used.

Chapter 4 includes implementation of the project. We start with preliminary assumptions and then we discuss the algorithms involved and then the actual implementation with screenshots of the execution.

Chapter 5 will portray the UML diagrams involved in the development process of the software.

In Chapter 6 we will discuss about future work and we will provide a conclusion of the project.

Chapter 2

Basic Information and Background

2. 1 Search Engines

Search engines have been used for many years. A search engine is a software script that searches documents and files for keywords and returns the results of the files that contain the keywords. Because search engines contain large amount of data it is important to display the results depending on the importance and relevance. This is determined by using various algorithms.

The goal of a search engine is to return the most relevant data so that the users return to do another search. Search engines assume that more popular a site is more valuable information it contains. This assumption has been proven right in terms of users.

There are three basic stages for a search engine: crawling – where content is discovered, indexing – where it is analyzed and stored in databases, and retrieval – where the user query fetches a list of relevant pages. We are concentrating on the retrieval aspects of the search engine.

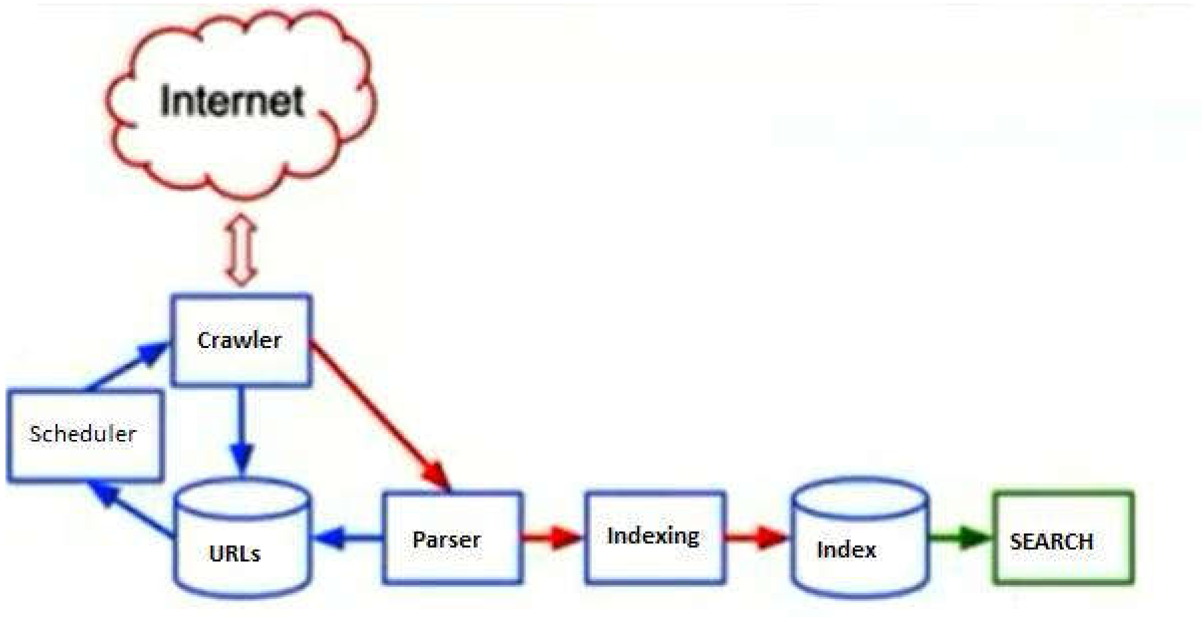


Figure 1 (http://searchengineland.com/how-search-engines-work-really)

When a user queries a search engine for information, the user actually searches through the indices created by the search engine. These indices are nothing but huge databases of information that is collected and stored and subsequently searched.

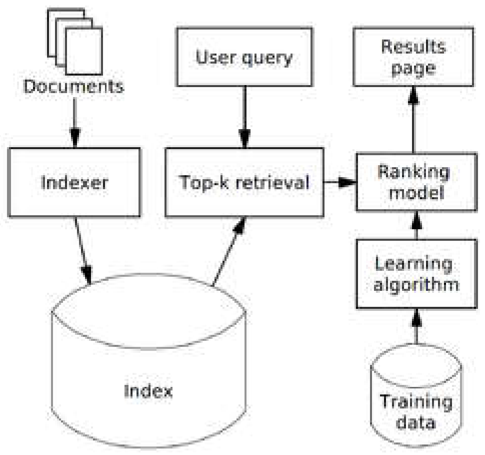


Figure 2

There are a lot of new issues we have to consider before creating a search engine.

- Suggestive Prediction
- Decision Management
- Iteration
- Mining for URLs
- Managing recent searches
- Relevancy of results

2.2 Purpose of efficient search

Large websites (University website, Online Shopping portals) have a huge database with a large set of keywords. An efficiently search and relevancy of displayed results can increase a huge amount of sales for a shopping website or give more information to a prospective student which is accurate and relevant to their search criteria, along with suggesting queries which are closer to the users search criteria.

2.3 Scope

The main goal of this project is to get the most relevant answers for a query in a least time possible using a concoction of instant, fuzzy search and proximity ranking applied on a database.

The proximity of matching keywords in answers is an important metric to determine the relevance of the answers. Search queries typically contain correlated keywords, and answers that have these keywords together are more likely what the user is searching for.

2.4 Existing System

Existing methods only identify a single tuple unit to answer keyword queries. However, they neglect the fact that in many cases, we need to integrate multiple related tuple units to answer a keyword query. To address this problem, we implement a structure-aware-index-based method to integrate multiple related tuple units to effectively answer keyword queries.

Disadvantage:

→ Single-keyword search without ranking

→ Keyword search without ranking

2.5 Instant Search

Many recent studies have been focused on instant search, also known as type-ahead search. The studies in proposed indexing and query techniques to support instant search. The studies in presented tree based techniques to tackle this problem.

When the user starts typing a query, the search engine offers suggestions before even finishing the type. One of the key principles in instant search is recognition over recall – it is similar to the notion that people are better at recognizing things they have previously experienced than they are at recalling from memory. As the user types into the search box, it tries to predict the query based on the characters that are entered.

Instant search does its best to remain unassertive, so users can still enter a query in full if they like. But selecting the prompts saves time and keystrokes.

2.6 Fuzzy Search

The studies on fuzzy search can be classified into two categories, gram-based approaches and trie-based approaches. In the former approach, sub-strings of the data are used for fuzzy string matching. The second class of approaches indexes the keywords as a trie, and relies on a traversal on the trie to find similar keywords. This approach is especially suitable for instant and fuzzy search since each query is a prefix and trie can support incremental computation efficiently.

A fuzzy search is done by means of a fuzzy matching program, which returns results based on likely relevance even though the words and spellings may not be correct. It operates like a spell-checker. The program can also find synonyms and related terms, working like a reference tool.

Superfluous results are likely to occur for terms with multiple meanings, only one of which is the result that the user desires.

A fuzzy matching program can correct common input typing errors, as well as errors introduced by optical character recognition scanning of printed documents. The program can return results with content that contains a word along with prefixes and suffixes. For example, if "form" is entered then the results returned can be of the pages that contain "format" or "formally".

2.7 Proximity Ranking

Proximity is highly correlated with document relevancy, and proximity aware ranking improves the precision of top results significantly. However, there are only a few studies that improve the query efficiency of proximity-aware search by using early-termination techniques exploited document structure to build a multi-tiered index to terminate the search process without processing all the tiers.

Proximity refers to the closeness between the keywords in a query. Closer the words are better the search is.

Proximity ranking is responsible for getting relevant answers to the user query in minimum time.

Chapter 3
Overview of the Solution

3.1 Proposed Solution

The efficient way to address the problem is by terminating the queries early that allows the search algorithm to find top answers without generating all the answers for the query (It only generates top answers). The idea is to traverse the inverted index of a given data file or a data set in an order and stop the probing once the most relevant results are in the solution set.

The order which is being used for the traversal of the index is very critical for early termination. We will be using an instant-Fuzzy search along with proximity ranking in early termination keywords.

3.2 Module Organization

The project consists of two parts.

The front end is a JSP web page which has two modules.

1. The admin: The admin for the search engine will have access to the database and can add keywords to the related web pages the admin can add or remove the files and documents from the database. He can also view the index recently made on the search engine.
2. The User: It with the user interface which will have a search bar which provides with instant searches on the go and will also give fuzzy search results if available for the query.

The back end is a Sql database which is connected using an Apache Tomcat which is responsible for connecting the database and the web page.

3.3 System Requirements

Hardware Interface

Processor	Pentium –IV
RAM	256 MB
Hard Disk	20 GB

Software Requirements

Operating System	Windows Family

Application Server	Tomcat5.0/6.X
Front End	HTML, Java, Jsp
Scripts	JavaScript
Server side Script	Java Server Pages.
Database	MySql
Database Connectivity	JDBC

3.4 Software Overview

JAVA

The programmer writes Java source code in a text editor which supports plain text. Normally the programmer uses an Integrated Development Environment (IDE) for programming. An IDE supports the programmer in the task of writing code, e.g. it provides auto-formatting of the source code, highlighting of the important keywords, etc.

At some point the programmer (or the IDE) calls the Java compiler (javac). The Java compiler creates the byte code instructions. . These instructions are stored in .class files and can be executed by the Java Virtual Machine.

J2SE

J2SE is a collection of Java Programming Language API (Application programming interface) that is very useful to many Java platform programs. It is derived from one of the most dynamic programming language known as "JAVA"

J2SE is a collection of Java Programming Language API (Application programming interface) that is very useful to many Java platform programs. It is derived from one of the most dynamic programming language known as "JAVA" and one of its three basic editions of Java known as Java standard edition being used for writing Applets and other web based applications.

J2SE platform has been developed under the Java umbrella and primarily used for writing applets and other Java-based applications. It is mostly used for individual computers. Applet is a type of fast-working subroutine of Java that is platform-independent but work within other frameworks. It is a mini application that performs a variety of functions, large and small, ordinary and dynamic, within the framework of larger applications.

J2SE provide the facility to users to see Flash movies or hear audio files by clicking on a Web page link. As the user clicks, page goes into the browser environment and begins the process of launching application-within-an-application to play the requested video or sound application. So many online games are being developed on J2SE. JavaBeans can also be developed by using J2SE.

J2EE

Java 2 Platform Enterprise Edition. J2EE is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitier, Web-based applications.

JDBC

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

- Making a connection to a database
- Creating SQL or MySQL statements
- Executing that SQL or MySQL queries in the database
- Viewing & Modifying the resulting records

All of these different executables are able to use a JDBC driver to access a database and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

JAVA SCRIPT

Java Script is Netscape’s cross–platform, object-based scripting language for client server application. JavaScript is mainly used as a client side scripting language. This means that JavaScript code is written into an HTML page. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it. JavaScript can be used in other contexts than a Web browser. Netscape created server-side JavaScript as a CGI-language that can do roughly the same as Perl or ASP.

Fortunately most browsers can handle JavaScript nowadays, but of course some browsers do not support some bits of script.

Features of JavaScript (JS):

a. Browser interprets JavaScript.
b. JavaScript is object based and uses built-in, extensible objects and have no classes or inheritance
c. JavaScript is loosely typed language
d. In JavaScript object reference are checked at runtime
e. JavaScript is designed to supplement the capabilities of HTML with script that are capable of responding to web pages events. JSP has access to some extent of aspects of the web browser window.
f. JavaScript control browser and content but cannot draw graphics or perform networking.

Client side JavaScript features:

Client–side JavaScript has expressly been developed for use in a web browser in conjunction with HTML pages. This has certain consequences for security.

- JavaScript cannot read files from or write them to the file system on the computer. This would be a clear security hazard
- JavaScript cannot execute any other programs. This would also be unacceptable.
- JavaScript cannot establish any connection to whatever computer, except to download a new HTML page or to send mail. This, too, would create unacceptable hazards.

The Client-Side JavaScript also has the following features:

- Controls Document's appearance and content
- Control the browser
- Interact with the HTML forms
- Interact with the user
- Read and write client state with cookies

Server- Side JavaScript Features:

a. Embedded in HTML page
b. Executed at the server
c. Pre-complied for faster response
d. Access to Server-side objects
e. Encapsulation of the request

APACHE TOMCAT SERVER

Apache Tomcat is a web container developed at the Apache Software Foundation (ASF). Tomcat implements the servlet and the Java Server Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Tomcat includes its own HTTP server internally.

JAVA SERVER PAGES

Introduction:

Jsp technology enables you to mix regular static html with dynamically generated content from servlets. Separating the static html from the dynamic content provides a number of benefits over servlets alone.

Why use JSP:

Jsp is easy to learn and allows developers to quickly produce wed sites and application in an open and standard way. Jsp is based on java, an object-oriented language. Jsp offers a robust platform for web development.

Main reasons to Jsp:

a. Multi platform
b. Component reuse by using java beans and Ejb
c. Advantages if java

We can take one Jsp file and move it to another platform, web server or Jsp servlet engine.

Chapter 4
Implementation

4.1 Preliminary

Let R = {R1, R2 . . . Rn} be a set of records with text attributes, such as the tuples in a relational table or a collection of documents. Let D be the dictionary that includes all the keywords of R.

Query: A query q is a string that contains a list of keywords w1, w2, . . . , wi, separated by space. In an instant-search system, a query is submitted for each keystroke of a user. When a user types in a string character by character, each query is constructed by appending one character at the end of the previous query. The last keyword in the query represents the word currently being typed, and is treated as prefix, while the first l−1 keywords (w 1 , w 2 , . . . , w l−1) are complete keywords. (The proposed techniques can be extended to the case where each keyword in the query is treated as a prefix.) For instance, when a user types in "brain tumor" character by character, the system receives the following queries one by one: q1 = (b), q2 = (br, . . . ,) q10 = (brain, tumo), q11 = (brain, tumor).

Answers: A record r from the data set R is an answer to the query q if it satisfies the following conditions: (1) For 1 ≤ i ≤ l − 1, it has a word similar to w i , and (2) it has a keyword with a prefix similar to w l . The meaning of "similar to" will be explained shortly. For instance, r 1, r 3, and r 4 are answers to q = (heart, surge), because all of them contain the keyword "heart". In addition, they have words "surgery", "surgeons", and "surgery", respectively, each of which has a prefix similar to "surge".

Ranking: Each answer to a query is ranked based on its relevance to the query, which is defined based on various pieces of information such as the frequencies of query keywords in the record, and co-occurrence of some query keywords as a phrase in the record.

Basic Indexing: As the techniques described in Ji et al. that combines fuzzy and instant search, we use three indexes to answer queries efficiently, a trie, an inverted index, and a forward index. In particular, we build a trie for the terms in the dictionary D. Each path from the root to a leaf node in the trie corresponds to a unique term in D.

Top-k Query Answers: Given a positive integer k, we compute the k most relevant answers to a query. One way to compute these results is to first find all the results matching the query conditions, then rank them based on their score.

4.2 Concepts Involved

Indexing: Indexing collects parses and stores the data for use. It is the search engine index that provides the results for queries, and pages that are stored within the search engine that appear on the search engine results page. Without a search engine index, the search engine would take more time and effort each time a search query was initiated, as the search engine would have to search not only every web page or piece of data that has to do with the particular keyword used in the search query, but every other piece of information it has access to, to ensure that it is not missing something that has something to do with the particular keyword.

As new documents are appended to the index, no rebuilding of the index is required.

Inverted Index: To make the query process faster, it is easier to sort the index by terms. With the inverted index we only have to look up for the list of documents that contain the keyword in the query instead of scanning all the documents.

The inverted index is built by sorting the index according to the terms.

4.3 Algorithms:

Algorithm 1: We use incremental computation for searching valid phrases as the user types each letter in the search bar. A query is divided into qi and qj. qi is the query which is already entered by the user and query qj has all the possible combinations of valid phrases starting with qi.

If a user wants to search for "Computer Science Department" he starts typing with the letter "C". A tree is created for all the possible results starting with "C". A tree is pruned as we go along and each node represents a result which can be generated by the entered query.

As C is entered we suggest all the possible valid phrases starting with C.

As the user types his next word "CO", we go back to the stored cache and we compute the valid phrases starting with "CO".

The main advantage of **incremental computing** is that as we go along down the nodes (node n1), we will also have all the suggestions from other nodes (n2 & n3). Thus the cache phrases may not be used until a new phrase is entered by the user.

The algorithmic approach is as given below.

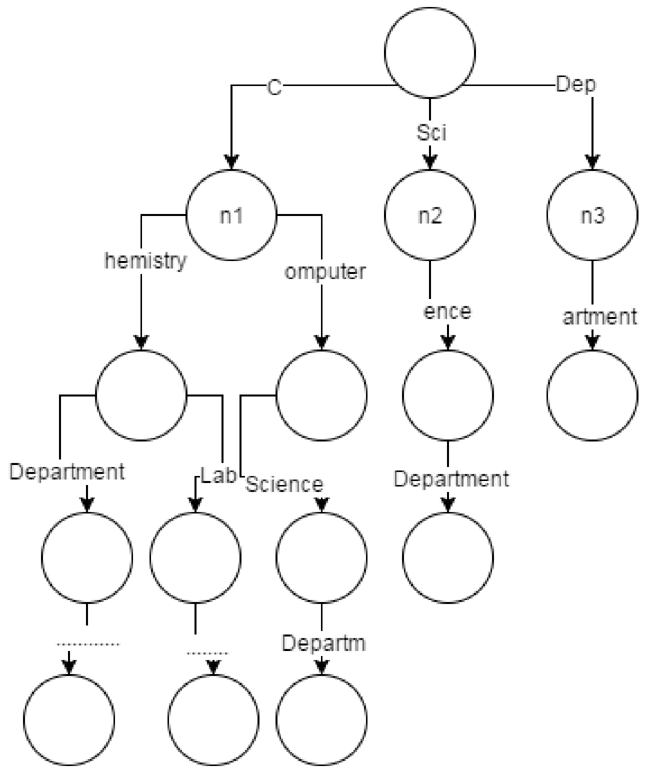


Figure 3

<u>Algorithm</u> Compute validphrases (q,C)

```
Input : query q = <w_1,w_2,……w_l>
        Keyword; a cache module C;
Output :a valid-phrase vector V;
(q_c,V_c) ← FindLongestCachedPrefix (q,C)
m ← number of keywords in q_c
if m > 0 then //cache hit
   for i←1 to m-1 do //copy the valid-phrase vector
     L V[i] ← V_c[i]
   if w_m ==q_c[m] then //The last keyword of q_c is a complete keyword in q
```

```
    L V[m] ← $V_c$[m]
 else //Incremental computation for the last keyword retrieved from cache
    V[m] ← ∅
     foreach (start, S) in $V_c$[m] do
      newS ← compute active nodes for $w_m$ incrementally from S
      if newS == ∅ then
         L V[m] ← V[m] U (start,newS)
foreach (start,S) in V[m] do //incremental computation for the phrases partially cached
     for j ← m+1 to $l$ do
      newS ← compute active nodes from S by appending $w_j$
      if newS == ∅ then break
      V[j] ← V[j] U (start,newS)
      S ← newS
for i← m + 1 to $l$ do //computation of non-cached phrases
   S ← compute active nodes for $w_i$
   V[i] ← V[i] U (i,S)
   for j← i+1 to $l$ do
     newS ← compute active nodes from S by appending $w_j$
     if newS == ∅ then break
     V[j] ← V[j] U (I,newS)
    S ← newS
cache(q,V) in C
return V
```

*** This algorithm is used for instant search to generate results quickly and on the go.**

Algorithm 2: Segmentation of a query is used to **generate segments of each query and search the results of each segment individually and in combinations with other segments.**

A query "Computer Science Department" is divided into more than three segments as shown in the table below.

Segment 1	Computer Science Department
Segment 2	Computer | Science Department
Segment 3	Computer Science | Department
Segment 4	Computer | Science | Department

Segment 4 is the base case for the recursion, where the start position of the current phrase is the beginning of the query. We can convert this recursive algorithm into a top-down dynamic programming algorithm by memorizing all the computed results for each end position.

Each phrase has a start position and an end position in the query.

The start position is stored in V [end] along with its computed active-node set. If there is a segmentation for the query w1 , . . . , w start−1 , we can append the phrase [start, end] to it to obtain a segmentation for the query hw 1 , . . . , w end i. Therefore, to compute all the segmentations for the first j keywords, we can compute all the segmentations for the first i − 1 keywords.

This analysis helps us reduce the problem of generating segmentations for the query (wi, . . . ,wj) to solving the sub problems of generating segmentation for each query.

```
Algorithm GenerateSegmentations (q,V,end)
Input : a query with a list of keywords
            $q = \{w_1, w_2, \ldots w_l\}$ ;
            its valid phrase vector V ;
            a keyword position end (end <= $l$);
Output : a vector $P_{end}$ of all valid segmentations of $w_1, w_2, \ldots w_{end}$
$P_{end} \leftarrow \emptyset$
foreach (start,$S_{start,end}$) in V[end] do
  if start == 1 then //Base case
  L $P_{end} \leftarrow P_{end} \cup \langle w_{start} \ldots w_{end} \rangle$
  else
```

```
    foreach seg in
    GenerateSegmentations (q,V,start-1)
     do
    seg ← seg | <$w_{start}$……$w_{end}$>
    $P_{end}$ ← $P_{end}$ U seg
return $P_{end}$
```

Ranking Segmentation:

The final query plan is an order of segmentations that are to be executed. The query plan needs to rank the segmentations.

Segmentation ranking relies on a comparator. This comparator compares the segmentations based on: (1) the summation of minimum edit distance between each valid phrase in the segmentation and its active nodes and (2) the number of phrases. The segmentation that has the smaller minimum edit distance is ranked higher. If two segmentations have same edit distance, then the segmentation with fewer segments has a higher rank.

When there are fewer phrases in segmentation, the number of keywords in a phrase increases. Having more keywords in a phrase can result in better answers because more keywords have proximity in the answers.

Chapter 5

End User View: We used a website created in JSP. The admin can upload new data, or add keywords to the existing database. The user has to login to access the search bar. When the user types for the queries in the search bar we use instant search approach to suggest phrases on the go. After the user hits search button we provide the user with fuzzy search and proximity ranking results. The index is stored and can be viewed in admin mode.

5.1 UML Diagrams:

Class Diagram

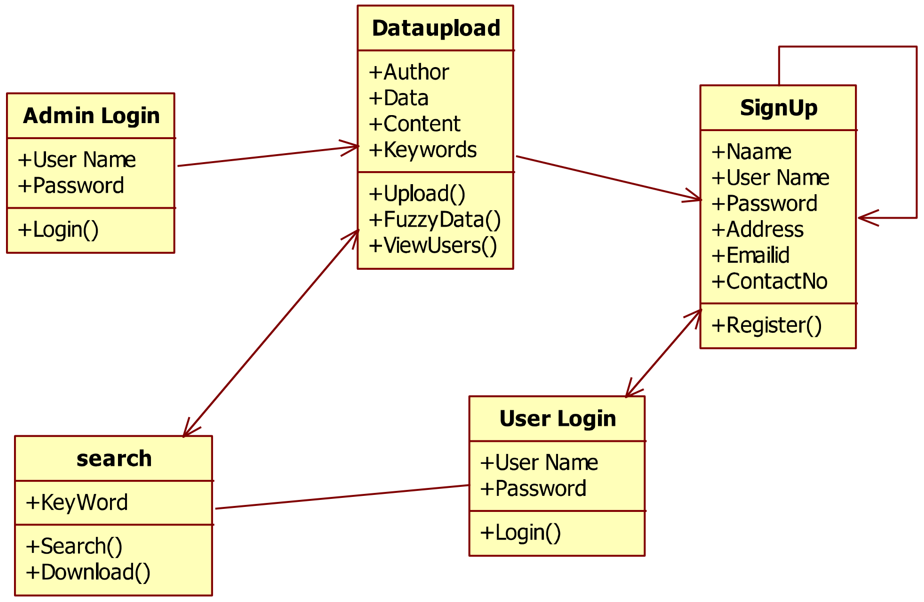


Figure 4

Use Case Diagram

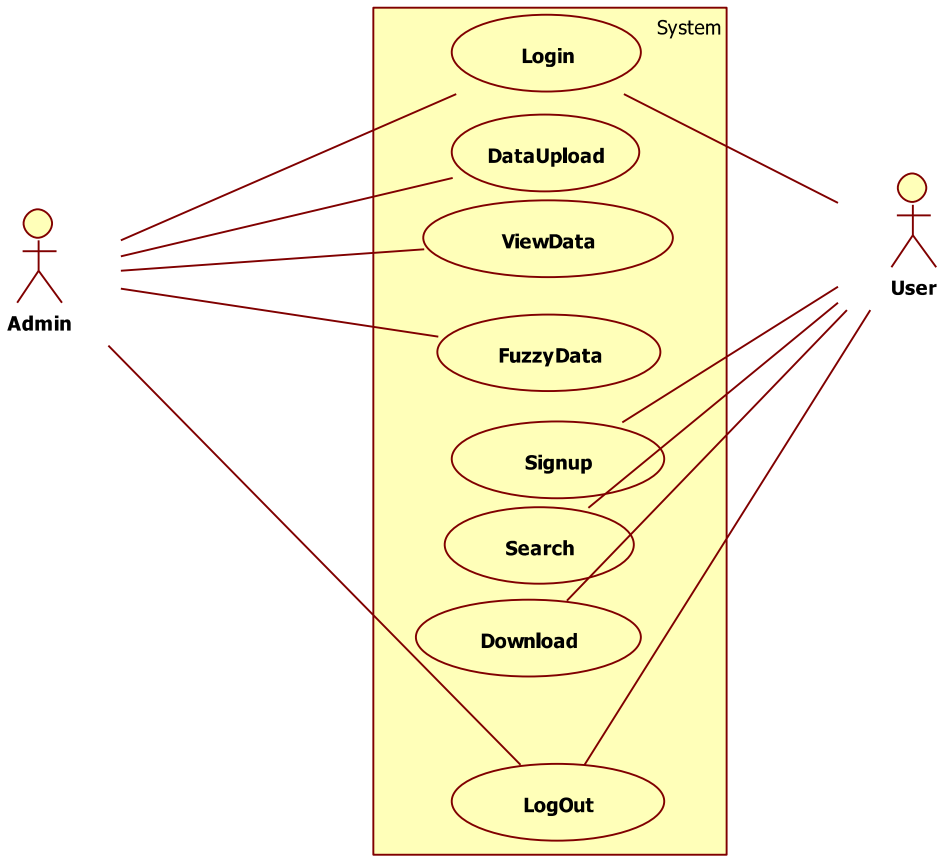


Figure 5

Sequence Diagram

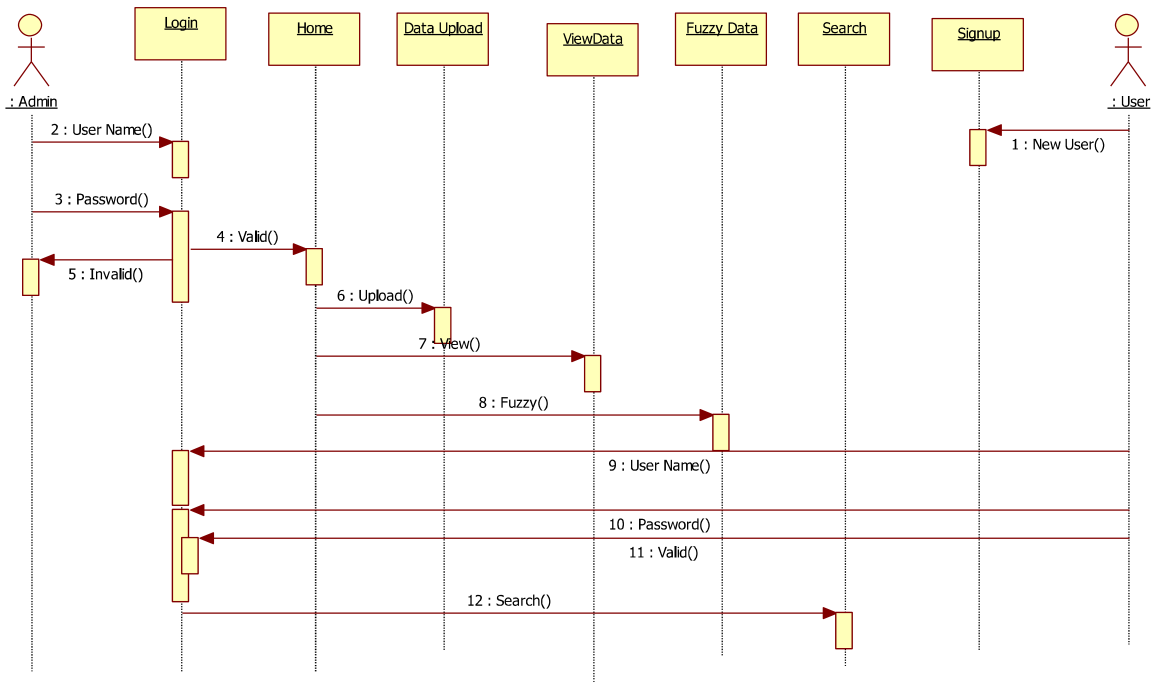


Figure 6

Deployment Diagram

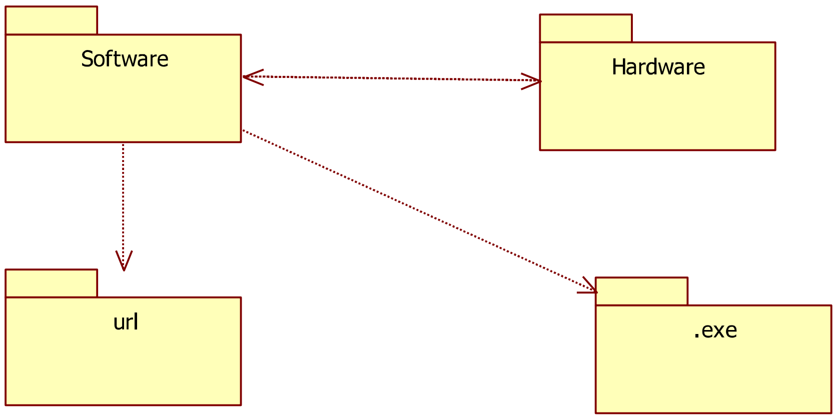


Figure 7

Activity Diagram

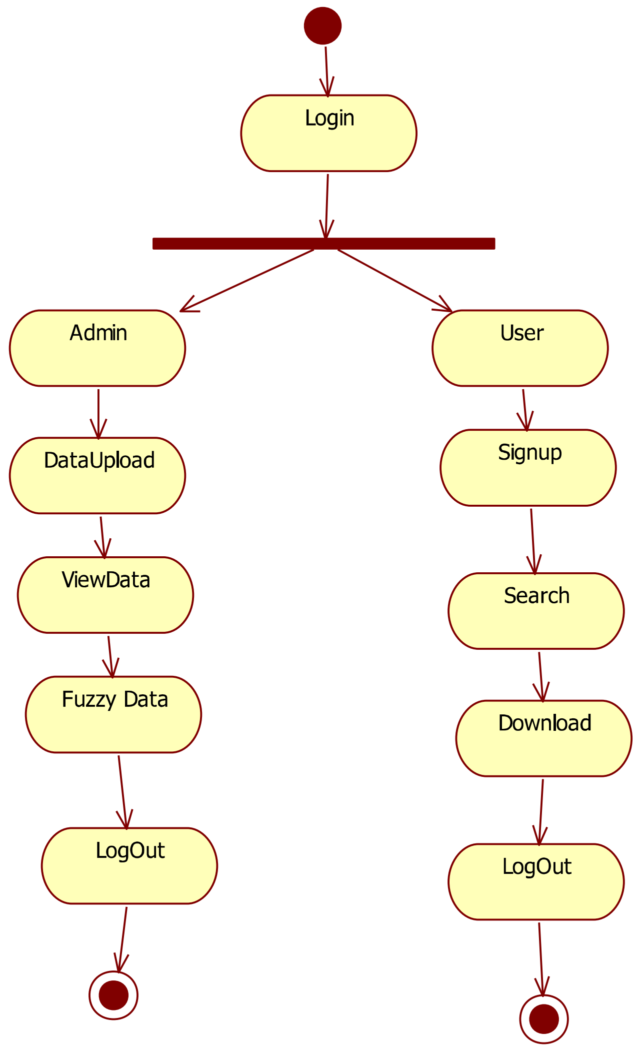


Figure 8

Component Diagram

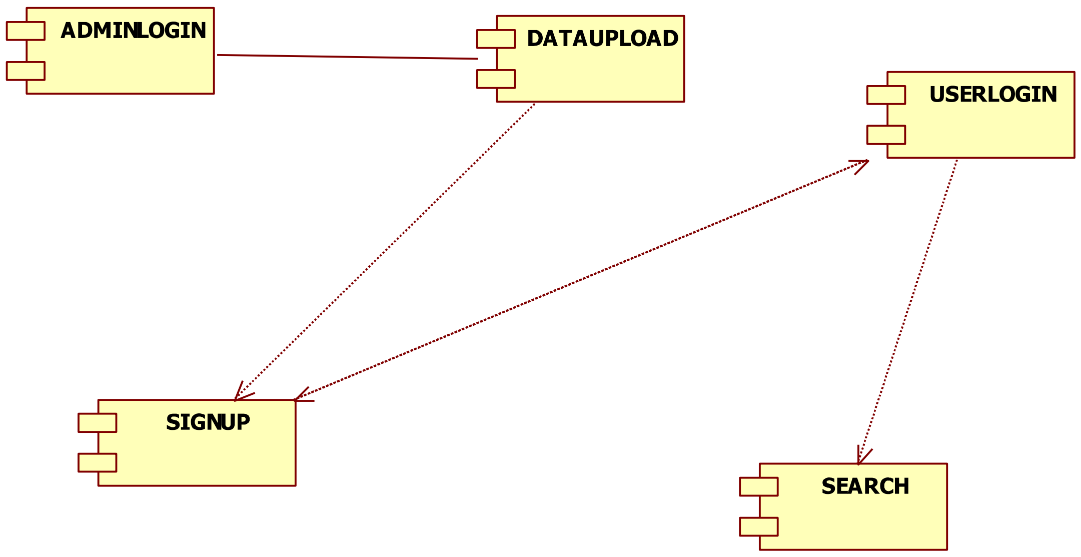


Figure 9

Collaboration

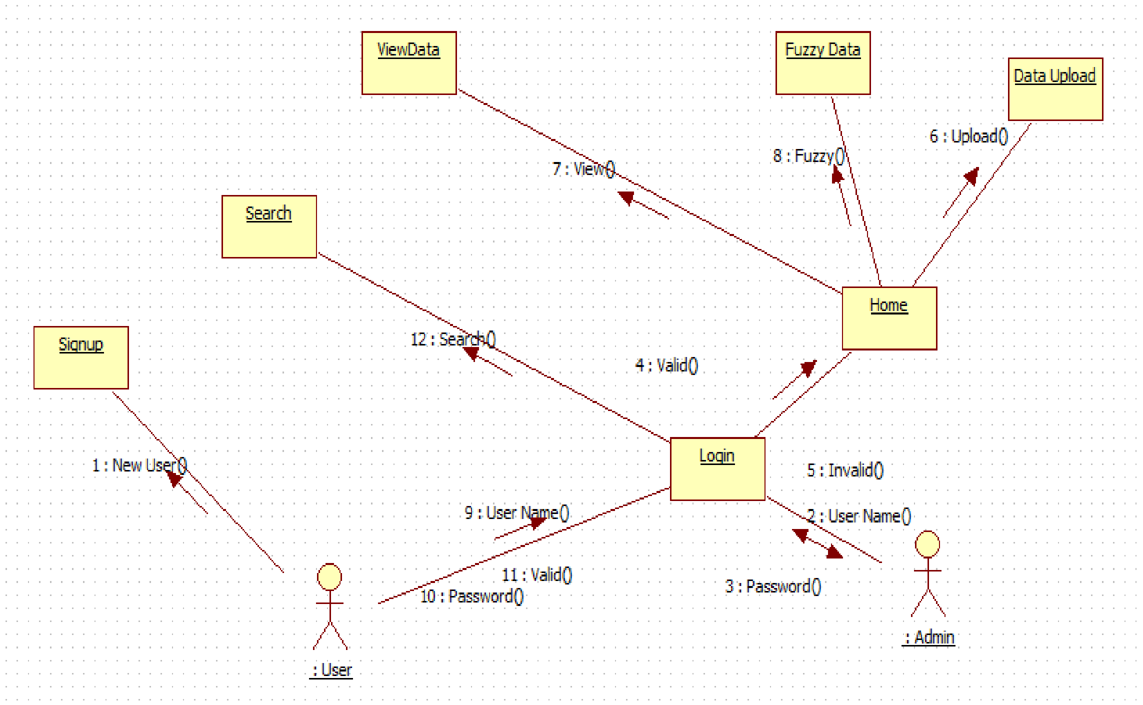


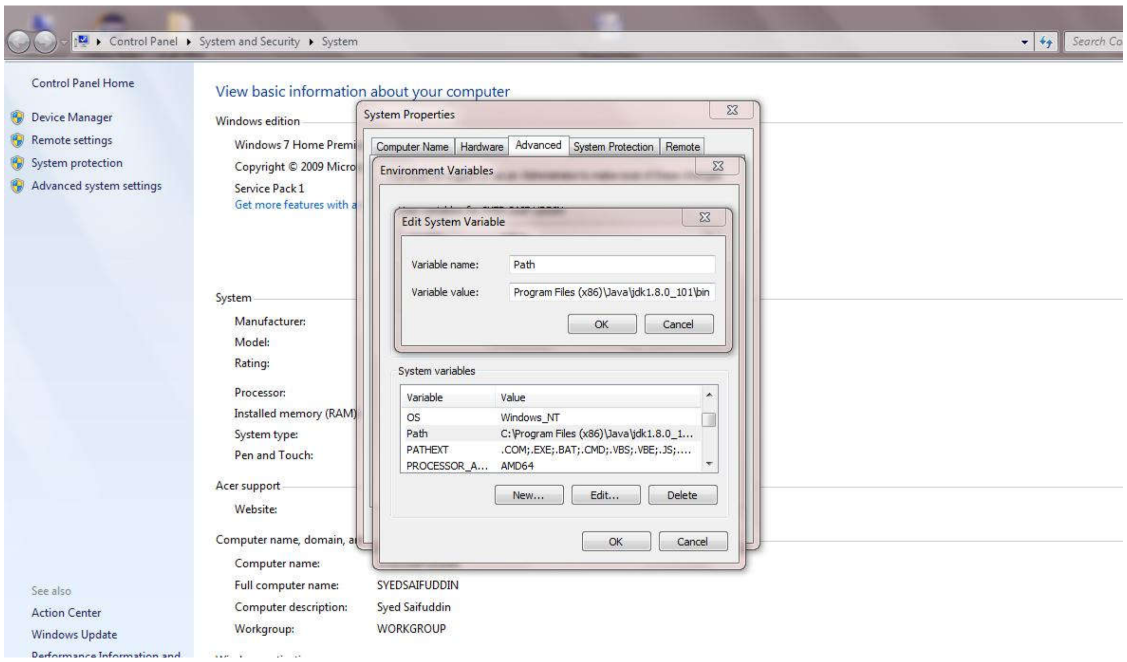
Figure 10

The following software components are being used in our project:

1. Apache Tomcat Server
2. SqlYog (Sql Server)
3. Tomcat Daemon
4. JDK 12
5. Web Browser

Step 1. Install JDK and set path

In this step the latest JDK is installed on windows 7 and then the variable path is set where the JDK is installed. Other configurations are being made to run the machine as desired.



Windows edition
Windows 7 Home Premi
Copyright © 2009 Micro
Service Pack 1

Variable name: Path
Variable value: Program Files (x86)\Java\jdk1.8.0_101\bin

System variables

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\Java\jdk1.8.0_1...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

System
Manufacturer:
Model:
Rating:
Processor:
Installed memory (RAM):
System type:
Pen and Touch:

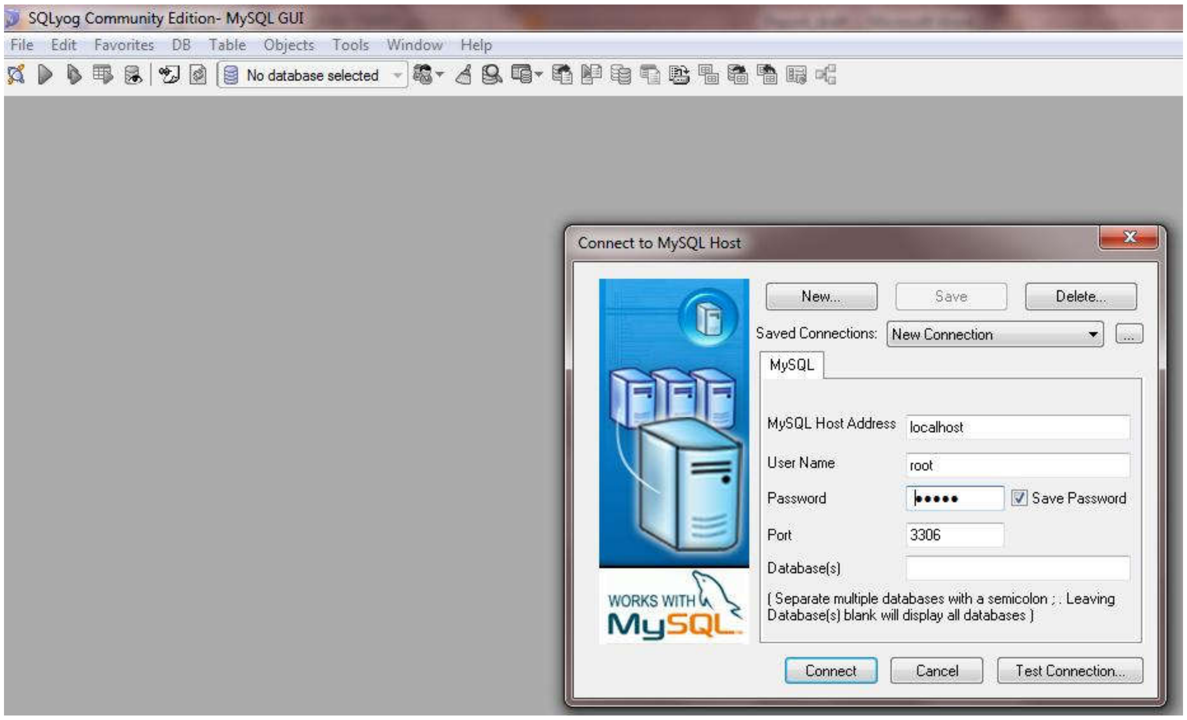
Acer support
Website:

Computer name, domain, a
Computer name:
Full computer name: SYEDSAIFUDDIN
Computer description: Syed Saifuddin
Workgroup: WORKGROUP

Figure 11

Step 2. Install Apache Tomcat and SqlYog

Apache tomcat and SqlYog are installed on the Windows 7 system on desired configurations. The port number to SQL will be the same as mentioned in configuration files.

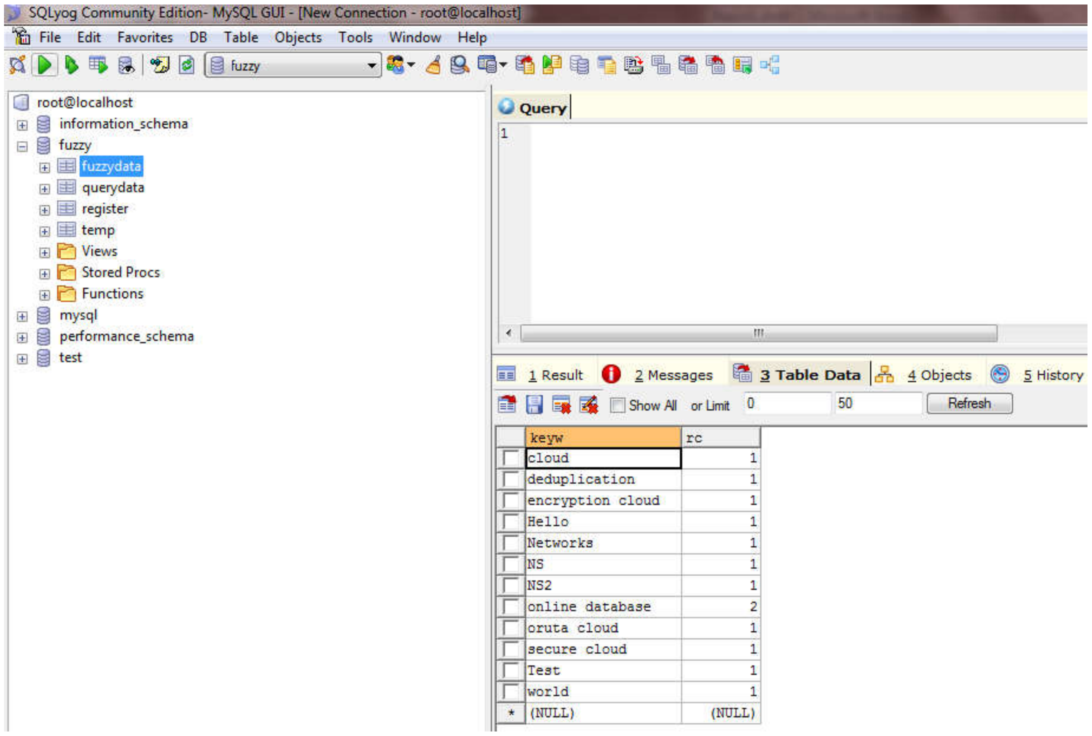

SQLyog Community Edition- MySQL GUI
File Edit Favorites DB Table Objects Tools Window Help
No database selected
Connect to MySQL Host
New... Save Delete...
WORKS WITH MySQL
MySQL
Connect Cancel Test Connection...

Saved Connections: New Connection

MySQL Host Address: localhost
User Name: root
Password: •••••
[✓ Save Password]
Port: 3306
Database(s):
(Separate multiple databases with a semicolon ; . Leaving Database(s) blank will display all databases)

Figure 12

The database file is initially empty, after entering several documents the folder in Sql will be populated.

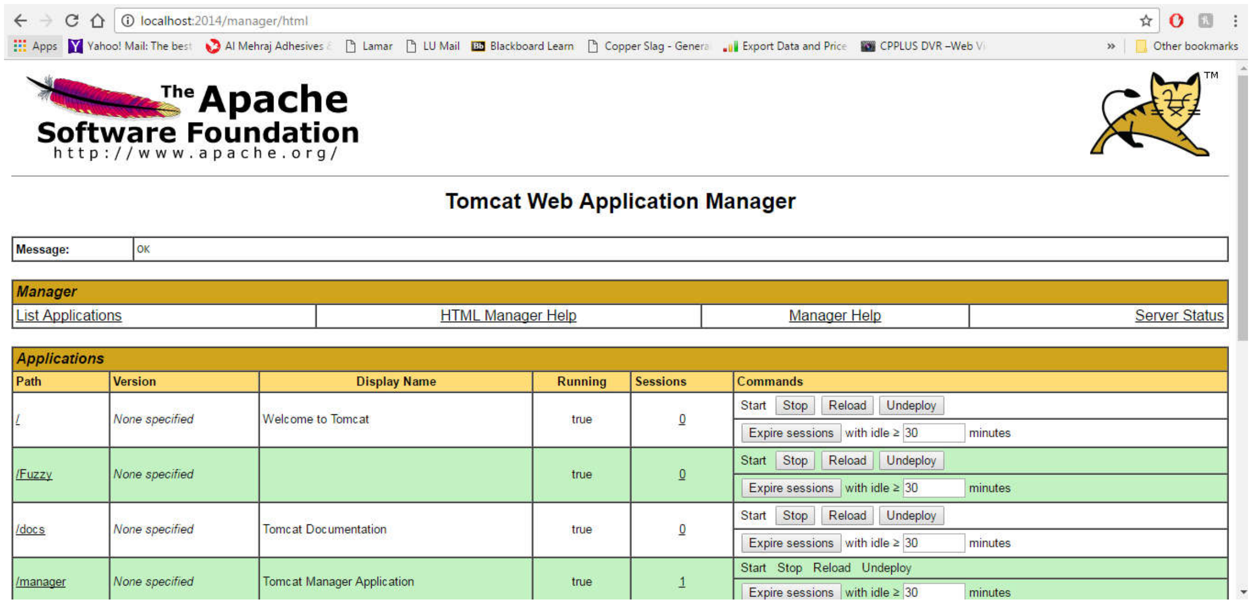


keyw	rc
cloud	1
deduplication	1
encryption cloud	1
Hello	1
Networks	1
NS	1
NS2	1
online database	2
oruta cloud	1
secure cloud	1
Test	1
world	1
(NULL)	(NULL)

Figure 13

The tomcat server is installed and set to desired configuration.

The webpages can be accessed at localhost:2014 (User Defined).



localhost:2014/manager/html

The Apache Software Foundation http://www.apache.org/

Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

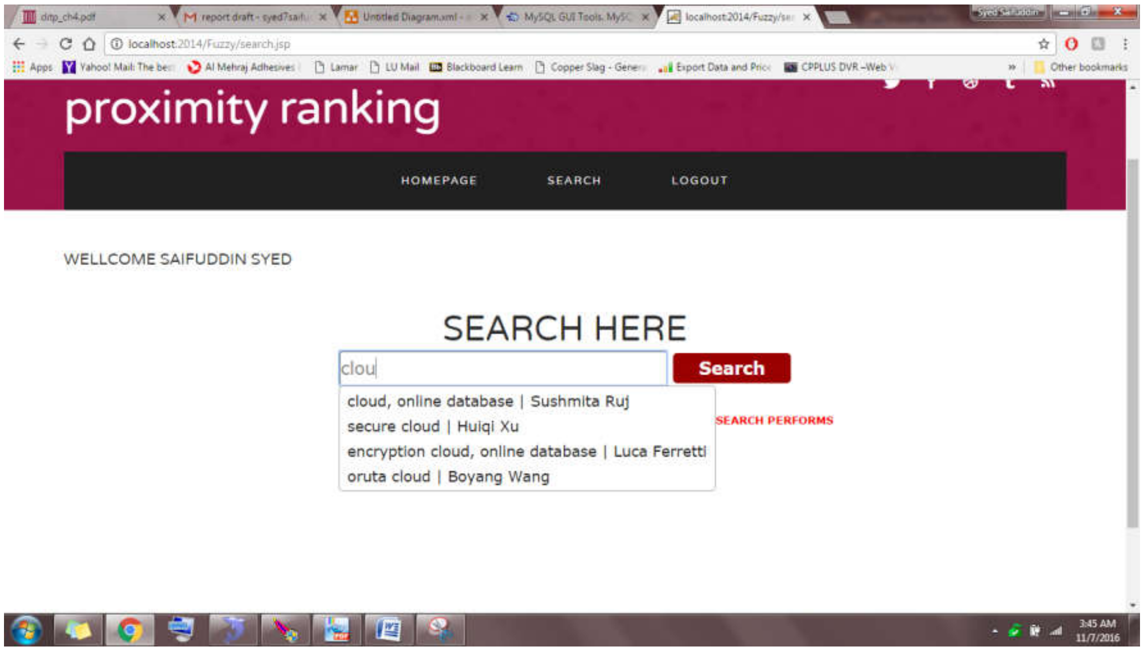
Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/Fuzzy	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 14

Step 3: Log into the user account and search queries in the database

Log into the user account using the credentials and click on the search tab. A search bar will appear. Click on the search bar and type any query the user wants to search from the database.



localhost:2014/Fuzzy/search.jsp

proximity ranking

WELLCOME SAIFUDDIN SYED

SEARCH HERE

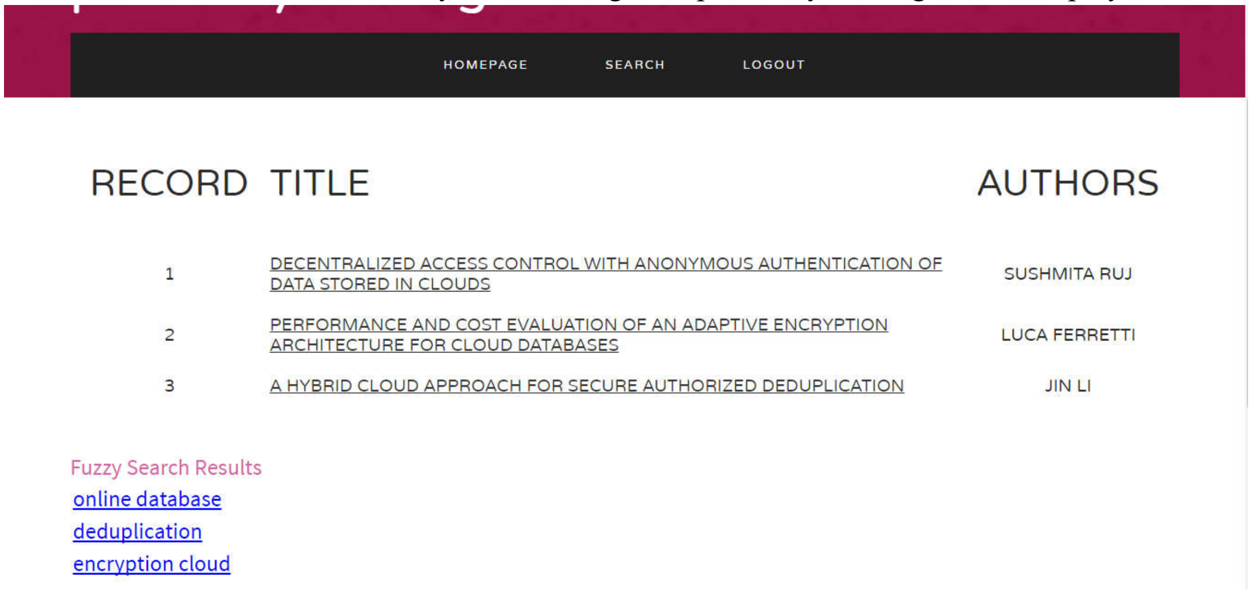
clou

cloud, online database | Sushmita Ruj
secure cloud | Huiqi Xu
encryption cloud, online database | Luca Ferretti
oruta cloud | Boyang Wang

SEARCH PERFORMS

Figure 15

The instant search results will be displayed as the user enters query alphabet by alphabet. After the user clicks enter the fuzzy results along with proximity ranking will be displayed.



RECORD	TITLE	AUTHORS
1	DECENTRALIZED ACCESS CONTROL WITH ANONYMOUS AUTHENTICATION OF DATA STORED IN CLOUDS	SUSHMITA RUJ
2	PERFORMANCE AND COST EVALUATION OF AN ADAPTIVE ENCRYPTION ARCHITECTURE FOR CLOUD DATABASES	LUCA FERRETTI
3	A HYBRID CLOUD APPROACH FOR SECURE AUTHORIZED DEDUPLICATION	JIN LI

Fuzzy Search Results
online database
deduplication
encryption cloud

Figure 16

The fuzzy search result will also give search results in which the partial keyword is present as relevant search based on the ranking segmentation algorithm.

Chapter 6
Conclusion & Future Work

6.1 Conclusion:

In this project we studied how to improve ranking of an instant-fuzzy search system by considering proximity information when we need to compute top-k answers. We studied how to adapt existing solutions to solve this problem, including computing all answers, doing early termination. We compared our techniques to the instant fuzzy adaptations of basic approaches. In particular, our experiments on data showed the efficiency of the proposed technique for 2-keyword and 3-keyword queries that are common in search applications. We concluded that computing all the answers for the other queries would give the best performance and satisfy the high-efficiency requirement of instant search.

6.2 Future Work

The current approach is a very simplified and a straight forward method to rank and provide results for the search queries as the user enters the keywords. Most of the keywords have to be entered for each document to be displayed in the search result i.e. the dictionary is manually entered which is not possible for high volume of data.

In order to get more accurate results we propose to use a self learning algorithm which can store and update the dictionary as the keywords are searched again and again. This allows the users to get more accurate and frequently used search results which are more likely to be searched.

References

[1] I. Cetindil, J. Esmaelnezhad, C. Li, and D. Newman, “Analysis of instant search query logs,” in WebDB, 2012, pp. 7–12.

[2] R. B. Miller, “Response time in man-computer conversational transactions,” in Proceedings of the December 9-11, 1968, fall joint computer conference, part I, ser. AFIPS ’68 (Fall, part I). New York, NY, USA: ACM, 1968, pp. 267–277. [Online]. Available: http://doi.acm.org/10.1145/1476589.1476628

[3] C. Silverstein, M. R. Henzinger, H. Marais, and M. Moricz, “Analysis of a very large web search engine query log,” SIGIR Forum, vol. 33, no. 1, pp. 6–12, 1999.

[4] G. Li, J. Wang, C. Li, and J. Feng, “Supporting efficient top-k queries in type-ahead search,” in SIGIR, 2012, pp. 355–364.

[5] R. Schenkel, A. Broschart, S. won Hwang, M. Theobald, and G. Weikum, “Efficient text proximity search,” in SPIRE, 2007, pp. 287– 299.

[6] H. Yan, S. Shi, F. Zhang, T. Suel, and J.-R. Wen, “Efficient term proximity search with term-pair indexes,” in CIKM, 2010, pp. 1229– 1238.

[7] M. Zhu, S. Shi, N. Yu, and J.-R. Wen, “Can phrase indexing help to process non-phrase queries?” in CIKM, 2008, pp. 679–688.

[8] A. Jain and M. Pennacchiotti, “Open entity extraction from web search query logs,” in COLING, 2010, pp. 510–518.

[9] K. Grabski and T. Scheffer, “Sentence completion,” in SIGIR, 2004, pp. 433–439.

[10] A. Nandi and H. V. Jagadish, “Effective phrase prediction,” in VLDB, 2007, pp. 219–230.