IJDCST @October Issue- V-1, I-7, SW-9
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Efficient Data Searching Using Forward Search

[1]Neelamraju Gunja, [2]R. Vasantha
[1]Student, PG Scholar, Dept. of CSE, EVMCET,JONNALAGADDA, NARASARAOPET AP, India
[2] Asst Professor, Dept. of CSE, EVMCET,JONNALAGADDA, NARASARAOPET AP, India

Abstract: Internet search engines are much popularized keyword search paradigm. In conventional search systems on data submits it to the system and retrieves relevant information, when a user composes a query with different keywords. If the user doesn't know how to issue queries, he tries multiple queries and sees what the result is. A new information retrieving system that searches data as the user types in query keywords. We study fuzzy type-ahead search in XML data that the system searches XML data on the fly as the user types in query keywords. It allows users to discover data as they type, even if there is an error in query keywords. Our proposed method has the following features: a) Search as you type: It extends Auto complete by supporting queries with multiple keywords in XML data b) Fuzzy: It can find high-quality answers that have keywords matching query keywords approximately c) Efficient: Our effective index structures and searching algorithms can achieve a very high interactive speed. The keyword search is alternative method to search in xml data, where user no needs to know about the knowledge of xml data and query languages. We study research challenges in this new search framework. The proposed effective index structures and top-k algorithms to achieve a high interactive speed. We examine effective ranking functions and early termination techniques to progressively identify the top-k relevant answers to achieves high search efficiency and result quality.

Index Terms: Search System, Index structures, Auto complete.

I. INTRODUCTION

Searching using keywords is a mostly used mechanism for querying data such as XML data. Keyword search is important in information systems. A keyword search looks for words anywhere in the record. The advantage of keyword search is its simplicity-users do not have to learn complex query language and can issue query without any knowledge about structure of xml document. One important advantage of keyword searching is it enables users to search information neither knowing a complex query language such as SQL. Traditional methods use query languages such as XPath and XQuery to query XML data. These methods are dominant but distant to non-expert users.

- These query languages are hard to understand for non-database users
- These languages require the queries to be posed against the different and the original database schemes.

Search system over XML data using keywords submits it to the system and retrieves relevant information from XML data. It requires the user to have definite information about the structure and content of the basic data repository. Xml was designed to transport and store data. Recently, the database research community has been studying challenges related to keyword search in XML data. One important advantage of keyword search is that it enables users to search information without knowing a complex query language such as XPath or XQuery. This information-access paradigm requires the user to have certain knowledge about the structure and content of the underlying data repository. Many systems are introducing various features to solve this problem. Commonly used methods is Auto complete that predicts a word or phrase that the user may type in based on the partial string the user has typed.

Complete-Search does not support approximate search that is it cannot allow minor errors between query keywords and answers. We studied fuzzy type-ahead search in textual documents.

II. LITERATURE SURVEY

Frequently used method is Auto complete that predicts phrase that the user may type in based on the unfinished string the user has typed. The problem with Auto complete is that the system treats a query as a single string if it consist multiple keywords.

One solution to this problem given by Bast and Weber is Complete Search in textual Documents that can find related answers by allowing keywords in query, come out at any places in the solution. Type-forward search can provide users immediate response as users type in keywords and it does not require users to type in entire keywords. The Type-forward search can help users browse the data that user save typing attempt and efficiently find the answers. Architecture for type-forward search as shown in the fig.1

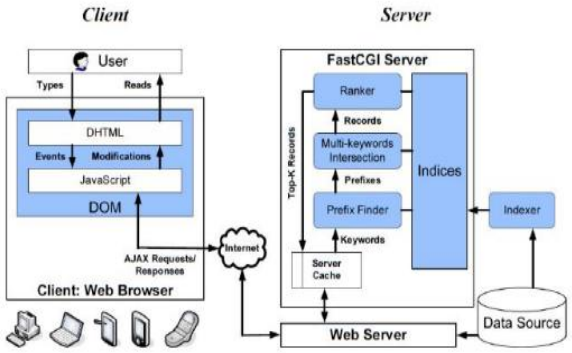

Fig. 1 Type-Forward Search System Architecture

We also considered type-forward search in relational databases. XML data in a type-forward search way and it is not inconsequential to expand existing techniques to support type-forward search in XML data because XML contains parent-child relationships. We need to identify appropriate XML sub trees that confine such structural relationships from XML data to answer queries with keywords. TFSX searches the XML data on the fly as user's type in query keywords even in the occurrence of small errors of their keywords. Each query with multiple keywords needs to be answered powerfully. The foremost challenge is search-effectiveness.

This short running-time requirement is mainly difficult when the backend repository has a huge amount of data. We suggest effective index structures and algorithms to solve keyword queries in XML data. Effective ranking functions and timely termination techniques to gradually discover top-k answers.

Notations:

In general XML document can be organized as a rooted tree with labeled nodes. Node v in the tree corresponds to an element in the XML document and has a label. Consider the XML document in Fig.2.

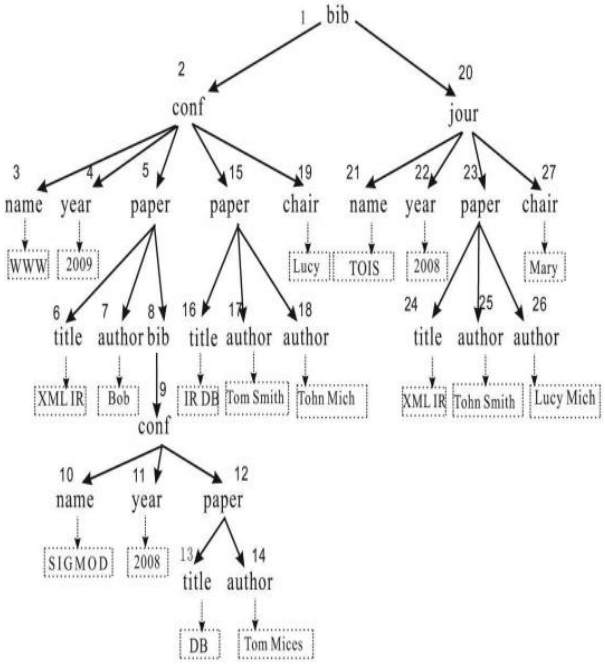

Fig. 2. An XML Document

A keyword query consists of a set of keywords $\{k_1, k_2 \ldots k_l\}$. We call the nodes in the tree that contain the keyword the content nodes for k_i. Ancestor nodes of the content nodes are called the quasi-content nodes of the keyword. Title is a content node for keyword "XML," and jour is a quasi-content node of keyword "XML."

Keyword Search in XML Data: There are many ways to identify the answers to a query on an XML

document. Frequently used one is based on the idea of lowest common ancestor. Given an XML document D and its XML nodes $v_1, v_2, \ldots, v_m$. We say a node u in the document is the lowest common ancestor of these nodes if for all ($1 \leq i \leq m$, $u \leq vi$) and there does not exist another node u` such that u < u` and u`≤ vi. The LCA-based algorithm primarily retrieves content nodes in XML data that contain the input keywords using inverted indices. The LCAs of the content nodes have the identities and takes the sub trees rooted at the LCAs as the answer to the query. To resolve the problem of use of LCAs as query solutions, We have various methods have been projected to progress search effectiveness and result quality. Exclusive lowest common ancestor is the one of the solution planned by Guo et al. and Xu and Papakonstantinou.

III. TYPE-FORWARD SEARCH IN XML DATA

We initially begin how TAFX works for multiple keyword queries in XML data, by allowing small errors of query keywords and inconsistencies in the data itself. There is an original XML document that resides on a server and a user accesses and searches the data through a web browser. Keystroke that the user types invoke a query consists existing string. Browser sends the query to the server that computes and profits to the user the best solutions ranked by their relevancy to the keyword query. Server primarily tokenizes the query into a number of keywords using delimiters such as the space character. The user may not type the entire keyword because keywords are taken as a partial keyword. We would like to know the feasible words the user intends to type for the partial keywords. We can only identify a set of complete words in the data set which have similar prefixes with the partial keywords.

Predicted words, sets of the complete words, we use edit distance to quantify the resemblance between two words. Calculate the distance between two words s1 and s2 is the lowest number of edit operations of single characters needed to transform the first one to the second. Server identifies the related sub trees in XML data for every input keyword that contain the predicted words. TAFX can save users time and efforts because they can find the answers even if they have not completed typing all the entire keywords or typing keywords with small errors.

TYPE-FORWARD SEARCH IN XML DATA:

Given an XML document D, the keyword query Q= { k1, k2,. . . kl} and an edit-distance threshold T. The predicted-word set be Wki = { w|w is a tokenized word in D and there exists a prefix of w, k`i, ed(ki , k`i) ≤ T }. For the keystroke that invokes Q, we return the top-k answers in RQ for a given value k. There are two challenges to maintain type-forward search in XML data. To interactively and powerfully identify the predicted words that have prefixes similar to the input partial keyword after each keystroke from the user. Second, o gradually and efficiently figure the top-k predicted answers of a query with multiple keywords, particularly when there are many predicted words.

IV. LCA-Based Fuzzy Type-Ahead Search

We use the semantics of ELCA to identify relevant answers on top of predicted words.

Index Structure:

We use a tries structure to index the words in the underlying XML data. Word w corresponds to a unique path from the root of the tries to a leaf node. Every node on the path has a label of a character in w. We store an inverted list of IDs of XML elements that contain the word of the leaf node. The tries structure for the tokenized words is shown in Fig. 3 with respective to fig.2.

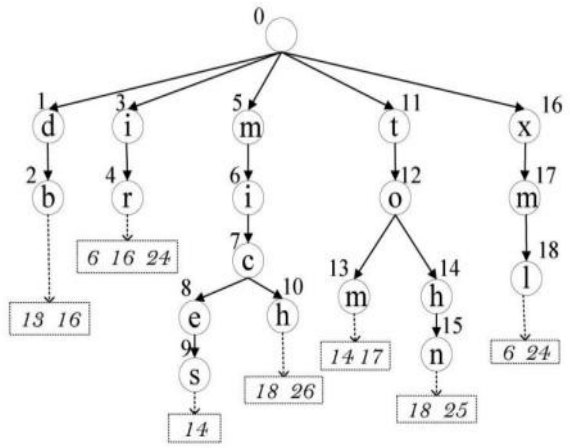

Fig. 3. The tries on top of words in Fig. 2

Answering Queries with a Single Keyword:
To answer a query with a single keyword using the tries structure. Every keystroke that a user types invokes a query of the current string and the client browser sends the query string to the server. We consider the case of exact search. The naive way to process such a query on the server is to answer the query from scratch as follows: we first find the tries node corresponding to this keyword by traversing the tries from the root. We locate the leaf descendants of this node and retrieve the corresponding predicted words and the predicted XML elements on the inverted lists. The server finds the tries node corresponding to this keyword. It locates the leaf descendants of node and retrieves the corresponding predicted words and the predicted XML elements. When the user types in the character "i," the client sends a query string "mi" to the server. Server answers the query from scratch as follows: it first finds node 6 for this string. It retrieves the corresponding predicted words. Other queries invoked by keystrokes are processed in a similar way. We can use a caching-based method to incrementally find the tries node for the input keyword. Each session keeps the keywords that the user has typed in the past and the corresponding tries node. The user may modify the previous query string arbitrarily or copy and paste a completely different string. We use this prefix to incrementally answer the new query by inserting the characters after the longest prefix of the new query one by one.

Answering Queries with Multiple Keywords:
We consider how to do fuzzy type-ahead search in the case of a query with multiple keyword. We first tokenize the query string into keywords $k_1,k_2,k_3 \ldots k_l$. We compute the corresponding active node k_i and for each such active node we retrieve its leaf descendants and corresponding inverted lists. we compute the predicted answers on top of lists Uk1, Uk2 . . . Ukl . We compute the predicted answers on top of lists Uk1, Uk2 . . . Ukl .

V. Progressive And Effective Top-K Fuzzy Type-Ahead Search

The LCA-based fuzzy type-ahead search algorithm in XML data has two main limitations. They use the "AND" semantics between input keywords of a query and ignore the answers that contain some of the query keywords. In order to compute the best results to a query, existing methods need find candidates first before ranking them, and this approach is not efficient for computing the best answers. We develop novel ranking techniques and efficient search algorithms. Each node on the XML tree could be potentially relevant to a keyword query and we use a ranking function to decide the best answers to the query. We index not only the content nodes for the keyword of the leaf node. But also those quasi-content nodes whose descendants contain the keyword.

Minimal-Cost Tree: we introduce a new framework to find relevant answers to a keyword query over an XML document. Each node on the XML tree is potentially relevant to the query with different scores. We define each node with its corresponding answer to the query as its sub tree with paths to nodes that include the query keywords, which referred as minimal-cost tree. Different nodes correspond to different answers to the query, and we will study how to quantify the relevance of each answer to the query for ranking.

Ranking Minimal-Cost Trees:

We first introduce a ranking function for exact search and then extend the ranking function to

support fuzzy search. To rank a minimal-cost tree, we evaluate the relevance between the root node and each input keyword and then combine these relevance scores for every input keyword as the overall score of the minimal-cost tree. The proposed two ranking functions to compute the relevance score between the root note n to an input keyword ki. Else, considers the case that n does not contain ki but has a descendant containing ki. Ranking method models each node n as a document that includes the terms contained in the tag name or text values of n. To address this issue, we extend the first ranking function and propose the second ranking function.

The distance between two points can indicate how relevant the node n is to keyword kj. Smaller the distance between n and p, the larger relevancy score between n and kj should be. We proposed the second ranking function to compute the relevance between n and kj as follows:

$$\mathrm{SCORE}_2(n, k_j) = \sum_{p \in P} \alpha^{\delta(n,p)} * \mathrm{SCORE}_1(p, k_j)$$

As the distance between n and p increases, n becomes less relevant to kj.

VI. TENTATIVE STUDY

We have implemented our method on real applications using our proposed techniques. We evaluate query results by human judgement. As XMark data set captures more complicated structures than DBLP data set. We implemented the hybrid algorithm of XRANK for the LCA-based method. We implemented XRANK's ranking functions. The server was running a program implemented in C++ and compiled with the GNU C++ compiler. The quality of the result can be assessed by LCA-based method and MCT-based methods. The quality of the result can be assessed by LCA-based method and MCT-based methods. Our MCT based search method achieves much higher resu1t quality than the LCA based method. The result analysis is as shown in the fig.4.

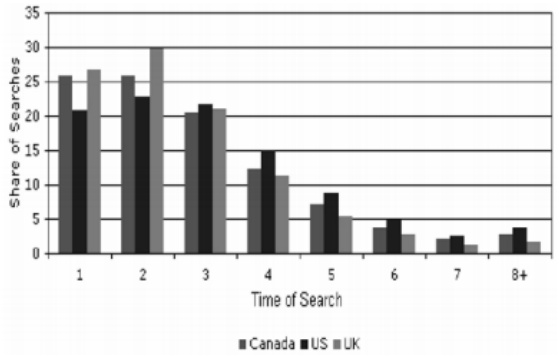

Fig.4. Number of Keywords in Search Terms by Country and Time

We evaluated the efficiency of computing the prefixes on the tree that are similar to a query keyword.

VII. CONCLUSION

We evaluated the efficiency of computing the prefixes on the tree that are similar to a query keyword. The proposed efficient incremental algorithm to respond single-keyword queries that are treated as prefix conditions. considered different algorithms for computing the answers to a query with multiple keywords. Well-organized algorithms are developed for incrementally computing answers to queries by using cached results of prior queries in order to get a high interactive speed on huge data sets. The LCA-based method to interactively discover the predicted answers and developed a minimal cost tree based search method to capably and step-by-step recognize the nearly all relevant answers. We devised a forward-index structure to further improve search performance.

VIII. REFERENCE's

[1]. Jianhua Feng, Guoliang Li, "Efficient Fuzzy Type-Ahead Search in XML Data," Proc. IEEE Transactions on Knowledge and Data Engineering, VOL. 24, NO. 5, MAY 2012.

[2]. Supriya sivapuja, Sk. Mohiddin, S Srikanth Babu

, Srikar Babu S.V, "Efficient Searching on Data Using Forward Search " Proc. International Journal of Emerging Trends & Technology in Computer Science, Volume 2, Issue 2, March – April 2013.

[3]. H. Bast and I. Weber, "Type Less, Find More: Fast Autocompletion Search with a Succinct Index," Proc Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR), pp. 364-371, 2006.

[4]. H. Bast and I. Weber, "The Completesearch Engine: Interactive, Efficient, and towards Ir&db Integration," Proc. Biennial Conf. Innovative Data Systems Research (CIDR), pp. 88-95, 2007.

[5]. D. Harel and R.E. Tarjan, "Fast Algorithms for Finding nearest Common Ancestors," SIAM J. Computing, vol. 13, no. 2, pp. 338- 355, 1984.

[6]. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "Xrank: Ranked Keyword Search over Xml Documents," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 16-27, 2003.

[7]. Z. Liu and Y. Chen, "Identifying Meaningful Return Information for Xml Keyword Search," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 329-340, 2007.