International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

User Profile Based Client Side Instant Search Mechanism With Use of TLB Mechanism and Fuzzy Search

Rupali A. Ingale[1], J. L. Chaudhari[2]

[1]ME CSE-Final Year, BSIOTR Wagholi (Pune), Maharashtra, India

[2]Assistant Professor, CSE Department, BSIOTR Wagholi (Pune), Maharashtra, India

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.A fuzzy search is a process that locates Web pages that are likely to be relevant to a search argument even when the argument does not exactly correspond to the desired information. A fuzzy search is done by means of a fuzzy matching program, which returns a list of results based on likely relevance even though search argument words and spellings may not exactly match. Exact and highly relevant matches appear near the top of the list. The fuzzy search can be done by trie-based approaches. 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. we propose the*** fetch from TLB and PageTables . We are maintaining log for user search log and part of relevant retrieved contents in TLB like mechanism. When user enters query to search engine, we will extract keywords from the query, this will need ***to porters stemming*** algorithm***, stopword removal algorithm and K-means*** clustering algorithm with consideration of hop count.

Keywords: Insatant search,Fuzzy search,TLB, Proximity Ranking

1. Introduction

The fuzzy search can be done by trie-based approaches. The fuzzy reasoning, described as "If *X* is A then Y is B" is said to be simple and conforms to human language. However in cases where the system has multi inputs and multi outputs, one has to build the fuzzy reasoning rules in multi dimensional input and output spaces in order to describe the behavior of the system. It is considered that the difficulty is caused by a mismatch between the description of the fuzzy reasoning and the actual records of inference rules which humans have. Fuzzy searching is much more powerful than exact searching when used for research and investigation. Fuzzy searching is especially useful when researching unfamiliar, foreign-language, or sophisticated terms, the proper spellings of which are not widely known. Fuzzy searching can also be used to locate individuals based on incomplete or partially inaccurate identifying information.

A fuzzy matching program can operate like a spell checker and spelling-error corrector. For example, if a user types "Misissippi" into Yahoo or Google (both of which use fuzzy matching), a list of hits is returned along with the question, "Did you mean Mississippi?" Alternative spellings, and words that sound the same but are spelled differently,

Search engine helps user to locate information from large storage media of content.

We propose search engine to reduce the work load of server. User search query in web its collect all web pages from server i.e. linking open data. All open data are copied temporarily in local system because of that user does not depends on server to view next web page so it reduce the work load of server. Local data 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. In that, fuzzy search method further improves user search experiences by finding relevant answers in system and filtering keyword similar to query keyword. A main computational challenge in this paradigm is high speed requirement i.e. each query needs to be answered within seconds to achieve an instant response and a high query throughput. We overcome the space and time limitation, fuzzy search method improves the user to fetch relevant record in server it temporarily stored in PC and further record search can be done without depending on server.

2. Literature Survey

K. Grabski and T. Scheffer[1],Proposed the problem of predicting the succeeding words of an initial fragment of natural language text. This problem setting is motivated by applications that include repetitive tasks such as writing.emails in call centers or letters in an administrative environment, many resulting documents are to some degree governed by specific underlying patterns that can be learned. The benefit of an assistance system to the user depends on both the number of helpful suggestions and the number of unnecessary distractions that they experience. Performance metrics for other text learning problems do not match the idiosyncrasies of this problem, we therefore have to discuss an appropriate evaluation scheme. Generative N-gram language models provide a natural approach to the construction of sentence completion systems , instance-based learning can easily be applied to this problem.

A.Nandi and H. V. Jagadish proposed system on predicting queries. Many systems do prediction by treating a query with multiple keywords as a single prefix string. Therefore,

if a related suggestion has the query keywords but not consecutively, then this suggestion cannot be found

H. Bast and I. Weber proposed many indexing and query techniques to support instant search, Imagine a user of a search engine typing a query. Then with every letter being typed, we would like an instant display of completions of the last query word which would lead to good hits. At the same time, the best hits for any of these completions should be displayed. Known indexing data structures that apply to this problem either incur large processing times for a substantial class of queries, or they use a lot of space. We present a new indexing data structure that uses no more space than a state-of-the-art compressed inverted index, but that yields an order of magnitude faster query processing times. Even on the large TREC Terabyte collection, which comprises over 25 million documents, we achieve, on a single machine and with the index on disk, average response times of one tenth of a second. We have built a full-fledged, interactive search engine that realizes the proposed autocompletion feature combined with support for proximity search, semi-structured (XML) text, subword and phrase completion, and semantic tags.

S. Chaudhuri, V. Ganti, and R. Motwani, Proposed Detecting and eliminating fuzzy duplicates is a critical data cleaning task that is required by many applications. Fuzzy duplicates are multiple seemingly distinct tuples, which represent the same real-world entity. We propose two novel criteria that enable characterization of fuzzy duplicates more accurately than is possible with existing techniques. Using these criteria, we propose a novel framework for the fuzzy duplicate elimination problem. We show that solutions within the new framework result in better accuracy than earlier approaches. We present an efficient algorithm for solving instantiations within the framework. We evaluate it on real datasets to demonstrate the accuracy and scalability of our algorithm.

G. Li, J. Wang, C. Li, and J. Feng Proposed the approach is especially suitable for instant and fuzzy search since each query is a prefix and trie can support incremental computation efficiently.

Type-ahead search can on-the-fly find answers as a user types in a keyword query. A main challenge in this search paradigm is the high-efficiency requirement that queries must be answered within milliseconds. In this paper we study how to answer top-k queries in this paradigm, i.e., as a user types in a query letter by letter, we want to efficiently find the k best answers. Instead of inventing completely new algorithms from scratch, we study challenges when adopting existing top-k algorithms in the literature that heavily rely on two basic list-access methods: random access and sorted access. We present two algorithms to support random access efficiently. We develop novel techniques to support efficient sorted access using list pruning and materialization. We extend our techniques to support fuzzy type-ahead search which allows minor errors between query keywords and answers. We report our experimental results on several real large data sets to show that the proposed techniques can answer top-k queries efficiently in type-ahead search.

3. Proposed System

We are maintaining log for user searchlog and part of relevant retrieved contents in TLB like mechanism.When user enters query to search engine, we will extract keywords from the query, this will need following algorithms .

3.1 Porters Stemming Algorithm

Porters stemming algorithm used to remove suffixes from the keyword . For example, we need to find occurances of keyword play in a result. We will get different words like **plays, played, playing etc.** We cant find extact occurance of **play** without removing suffixes like ed, ing, s, sses, ation, ational etc.Thre are some exceptional cases like **Red** this keyword is ended with **ed** but we dont want to remove **ed** from the **Red.** For stemming with such kind of examples porter have defined algorithm which has 6 steps implementation.Porter have defined some rules to remove suffixes for example (V-Vowel , C-Consonent) ,
PLAYED RED
CCVCVC CVC
Count m= |VC pairs| ie. for played m=2 and for red m=1.Rule defined by porter is if (m>1 && str.endsWith("ed")) remove "ed" from str else dont remove "ed".

3.2 Stopword Removal Algorithm

We need to download stopwords dataset from internet which is easlily available on many websites.Compare the words with stopword database and remove stopword occurances for further computations.

3.3 K-Means clustering algorithm with consideration of hop count

K-means clustering algorithm clusters the user keyword log based on its number of occurances.High frequency cluster is maintained at TLB side. When user enters query it will firstly look in TLB, if he dont gets relevant results then it calls for the pages/ results from fuzzy search mechanism .

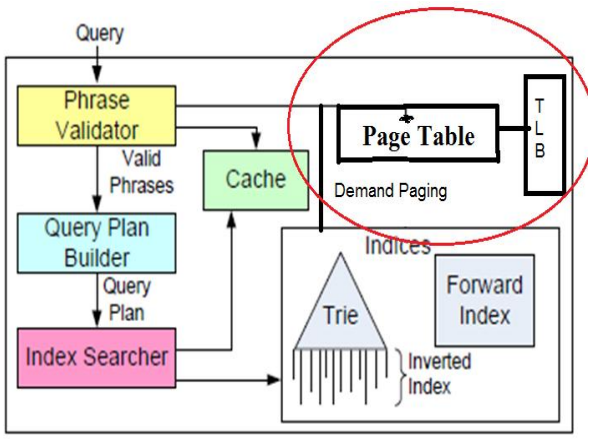
3.4 Fuzzy Search Mechanism

We formalize the problem of interactive, fuzzy search on a relational table, and our method can be adapted to textual documents, XML documents, and relational databases. Consider a relational table T with m attributes and n records. Let A = {a1, a2,...,am} denote the attribute set, R = {r1, r2,...,rn} denote the record set, and W = {w1, w2,...,wp} denote the distinct-word set in T. Given two words wi and wj , "wi wj" denotes that wi is a prefix string of wj .A query consists of a set of prefixes Q = {p1, p2,...,pl}. For each prefix pi, we want to find the set of prefixes from the data set that are similar to pi. 2 In this work we use edit distance to measure the similarity between two strings. The edit distance between two strings s1 and s2, denoted by ed(s1, s2), is the minimum number of edit operations (i.e., insertion, deletion, and substitution) of single characters needed to transform the first one to the second. For example, ed(smith, smyth) = 1. Definition 1 (Interactive Fuzzy Search). Given a set of records R, let W be the set of words

in R. Consider a query Q = {p1, p2,...,p} and an edit-distance threshold δ. For each pi, let Pi be {p i|∃w ∈ W, p i w and ed(p i, pi) ≤ δ}. Let the set of candidate records RQ be {r|r ∈ R, ∀1 ≤ i ≤ , ∃p i ∈ Pi and wi appears in r, p i wi}. The problem is to compute the best records in RQ ranked by their relevancy to Q. These records are computed incrementally as the user modifies the query, e.g., by typing in more letters.

3.4 Indexing

We use a trie to index the words in the relational table. Each word w in the table corresponds to a unique path from the root of the trie to a leaf node. Each node on the path has a label of a character in w. For simplicity, a node is mentioned interchangeably with its corresponding string in the remainder of the paper. Each leaf node has an inverted list of IDs of records that contain the corresponding word, with additional information such as the attribute in which the keyword appears and its position. For instance, Figure 3 shows a partial index structure for publication records. The word "vldb" has a trie node ID of 15, and its inverted list includes record IDs 6, 7, and 8. For simplicity, the figure only shows the record ids, without showing the additional information about attributes and positions.



Server architecture of instant-fuzzy search.

Figure 1: Proposed System

4. Conclusion

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, and indexing term pairs. We proposed a technique to index important phrases to avoid the large space overhead of indexing all word grams. We presented an incremental-computation algorithm for finding the indexed phrases in a query efficiently, and studied how to compute and rank the segmentations consisting of the indexed phrases. We compared our techniques to the instantfuzzy adaptations of basic approaches. We conducted a very thorough analysis by considering space, time, and relevancy tradeoffs of these approaches. In particular, our experiments on real 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.

References

[1] Cetindil∗, J. Esmaelnezhad,Taewoo Kim and Chen Li and Irvine "Efficient Instant-Fuzzy Search with Proximity Ranking",Available: http://www.imdb.com, as of July 2013.
[2] Cetindil, J. Esmaelnezhad, C. Li, and D. Newman, "Analysis of instant search query logs," in *WebDB*, 2012, pp. 7–12.
[3] 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.
[4] 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.
[5] G. Li, J. Wang, C. Li, and J. Feng, "Supporting efficient top-k queries in type-ahead search," in *SIGIR*, 2012, pp. 355–364.
[6] R. Schenkel, A. Broschart, S. won Hwang, M. Theobald, and G. Weikum, "Efficient text proximity search," in *SPIRE*, 2007, pp. 287–299.
[7] 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.
[8] 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.
[9] A. Jain and M. Pennacchiotti, "Open entity extraction from web search query logs," in *COLING*, 2010, pp. 510–518.