Client side instant search with use of TLB mechanism and Fuzzy Search

[1]Miss Mangala S. Teli, [2]Asst. Prof. Priti Subramanium
[1] PG Student (Computer Science &Engg.), S.S.G.B.C.O.E.T., Bhusawal
[2]Dept. of Computer Science &Engg., S.S.G.B.C.O.E.T., Bhusawal
India

***Abstract*: Searching anything is possible using search engine such as Google search engine. When user types query every user wants reply within short time. So Instant search is an information-retrieval method in which a system finds answers to a query instantly as a user types in character one-by-one. A fuzzy search in combination with instant search provides better results because some user doesn't know about exact keyword that time fuzzy search is useful. A fuzzy search check similarity between keyword to be found using fuzzy string matching algorithm and returns similar records not only exact. Exact and highly similar records appear from top of the list according to similarity. A trie-based approach is used for fuzzy search. A main computational challenge in fuzzy search 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 proposed make use of TLB and Page Table. Where multithreading is done to reduce retrieval time using TLB. The Levenshtein distance fuzzy string matching algorithm measures difference between two strings.**

Keywords—Fuzzy search, Instant search, TLB, Levenshtein Distance

I. INTRODUCTION

Trie-based approaches are used in fuzzy search. 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.

As compare to other searching techniques fuzzy searching is much more powerful when used for investigation and research area. Fuzzy searching is mostly useful when searching foreign-language, or sophisticated terms, unfamiliar and when the proper spellings are not widely known to user. Fuzzy searching can be used to locate individuals based on partially inaccurate or incomplete 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 proposed search engine to reduce searching time. There two datasets are used for experimental purpose. Firstly, dataset is divided into multiple pages and using TLB parallel searching is carried out over multiple pages i.e. multithreading. This reduced keyword searching time. While searching the Levenshtein distance algorithm is used to measure string difference using string metric. The Levenshtein distance is calculated as minimum number of single-character edits i.e. insertions, deletions or substitutions required to change one word into the other. This distance is used to find highly relevant records or top-k answers. We cannot find relevant answers by finding records with keywords matching the query exactly. This problem can be solved by supporting fuzzy search. Users often make typographical mistakes in their search queries. Meanwhile, small keyboards on mobile devices, lack of caution, or limited knowledge about the data can also cause mistakes.

Combining fuzzy search with instant search can provide an even better search experiences, especially for mobile-phone users, who often have the "fat fingers" problem; a main computational challenge in this search paradigm is its high- speed requirement. At the same time, compared to traditional search systems, instant search can result in more queries on the server since each keystroke can invoke a query, thus it requires a higher speed of the search process to meet the requirement of a high query throughput. What makes the computation even more challenging? One approach is to first find all the answers, compute the score of each answer based on a ranking function, sort them using the score, and return the top results. However, enumerating all these answers can be computationally expensive when these answers are too many. This case is more likely to happen compared to a traditional search system since query keywords in instant search are treated as prefixes and can have many completions. In addition, fuzzy search makes the situation even more challenging since there can be many keywords with a prefix similar to a query prefix.

II. LITERATURE SURVEY

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 [1]. H. Bast and I. Weber proposed many indexing and query techniques to support instant search [2]. M. Hadjieleftheriou and C. Li, K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, S. Chaudhuri, V. Ganti, and R. Motwani suggested former approach, sub-strings of the data are used for fuzzy string matching [3]. S. Ji, G. Li, C. Li, and J. Feng suggested approach which is especially suitable for instant and fuzzy search since each query is a prefix and trie can support incremental computation efficiently [4]. R. Fagin, A. Lotem, and M. Naor, F. Zhang, S. Shi, H. Yan, and J.-R.Wen explained extensively to support top-k queries efficiently [5,9]. G. Li, J. Wang, C. Li, and J. Feng adopted existing top-k algorithms to do instant-fuzzy search [6]. M. Persin, J. Zobel, and R. Sacks-Davis has proposed how inverted lists sorted by decreasing document frequency [7]. H. Yan, S. Shi, F. Zhang, T. Suel, and J.-R. Wen studied the effect of term-independent features in index reorganization [8].

I. Cetindil, J. Esmaelnezhad, C. Li, and D. Newman explained how to analyze query logs of instant-search systems. Instant-search system is compared with a traditional search system on the same data. The comparison shows that instant search can save typing effort by returning relevant answers before the user completes typing the query [10]. Silverstein presented an analysis of an AltaVista query log containing almost 1 billion entries. It is confirmed the conjecture that an average web user differs significantly from the user model assumed by the information retrieval com- munity. Surprisingly, for 85% of the queries only the first result screen is viewed, and 77% of the sessions contain only 1 query, i.e., the queries were not modified in these sessions. In the correlation analysis the queries of all users and found the strongest correlations resulted from short queries that were actually single-term phrase queries [11].

Most of the searching techniques use good ranking functions which are based on relevance score of query, query occurrences in datasets and distance between two query keywords in record. In proximity ranking the main focus is on exact query matching as user type query [1]. In instant search mainly three basic indexing techniques are used as inverted index, forward index and tries. Each query keyword is represented using tries. In tries each leaf node presents the inverted list of prefix and in forward index records id is represented for each record according to inverted list [8, 10].

Most of the searching techniques prefer top-k result. Firstly results matching to query condition are computed and secondly these results are ranked based on their relevance score. The main disadvantage of this technique is results matching to query keywords are many which reduce the system performance by increasing time to extract query. So it does not give the higher performance. This disadvantage is removed by using early termination technique.

A. Computing All Answers: A naive resolution is to initial calculate all the answers matching the keywords as follows. For every question keyword, we find the documents containing an identical keyword by computing the union of the inverted lists of those similar keywords. For the last question keyword, we have a tendency to contemplate the union of the inverted lists for the completions of every prefix the same as it. We have a tendency to see these union lists to search out all the candidate answers. Then we compute the score of every answer employing a ranking perform sort them supported the score, and come back the top-k answers.

B. Using Early Termination: To solve this problem, Li et al. [7] developed a technique that can find the most relevant answers without generating all the candidate answers. In this approach, the inverted list of a keyword is ordered based on the relevancy of the keyword to the records on the list. This order guarantees that more relevant records for a keyword are processed earlier.

C. Using Term-Pair Index: The intuition behind this strategy is that the computer program doesn't would like an excessive amount of time to method both terms though there's no term-pair list since inverted lists of these terms area unit comparatively short compared to those of alternative terms [5].

III. PROPOSED SYSTEM

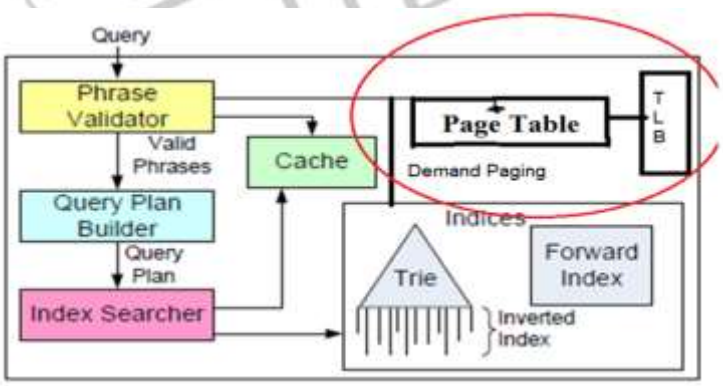

Fig. 6.1: Proposed architecture

- Server architecture of instant search:

In server architecture of instant fuzzy search, server receives requests and create inverted list of all query keywords present in dataset dictionary. Phrase validator mainly identifies the valid phrases present in dictionary. By comparing all matches and segmentations exact similar phrase is identified. Phrase validator computes valid phrases and based on trie structure inverted index is also computed. Query plan builder computes valid query plan which consists the valid segmentations with definite order. After query plan index searcher take each segmentation one by one and top-k results are computed. Cache module is used to store results computed by previous search and can be used to next query keyword search.

- Computation of valid phrases:

As algorithm described in [10], incrementally valid phrases are computed. Algorithm explains how active nodes are computed from inverted list and from cache, computing valid phrase incrementally based on cached valid phrase of previous queries. In this, query logs are used before searching query incrementally. If user has previously searched the query, search logs are created and searching results are stored in log. Next time when user searching same query, then firstly searcher search for log if that query present in log then get retrieved if not then according to incremental search that query is searched. Query log reduces searching time for those queries which are previously searched.

- Segmenting and ranking queries:

From incremental computation valid phrase vector is computed. From valid phrase vector all possible segmentations are generated by using divide and conquer algorithm [10]. Example, suppose query q=< Life is beautiful > then possible segmentations are present in following table.

1. "Life is beautiful"
2. "Life is | beautiful"
3. "Life | is | beautiful"

According to computed segmentations, these segmentations are ranked based on some ranking segmentations. Segmentations are ranked based on minimum edit distance and number of phrases present in segmentation. If any two segmentations have same minimum edit distance then they have same rank.

Dataset is divided into number of pages and TLB (Test load balancer) is used to divide process into number of subsets to be executed into parallel. The processes executed on same or different machine as different threads or processes. TLB decides which process needs to run in parallel across machines.

IV. EXPERIMENTAL RESULT

The performance of proposed techniques is on real data sets evaluated in this section. We used two data sets in the experiments, namely IMDB and Medline. Table I shows the Time Comparison between existing and proposed system against no of records. The IMDB data set from www.imdb.com/interfaces. The data stored in IMDB in the movies, actors, and characters tables, and constructed a table in which each record was a movie with a list of actors and a list of characters. Experiments carried out over 5000 records each dataset.

Table I: Time Comparison between existing and proposed system against no of records

No. of Records/System	Existing System Time(ms)	Proposed System Time(ms)
1000	1300	320
2000	1300	320
3000	1300	320
4000	1300	435
5000	1300	435

When no of datasets increases the computation time is also get increases. Cache hit for FA is better for all datasets. In 2- keyword search and 3-keyword search QA performs best, FA outperforms and TP mostly fail for 2 and more keyword search. From experimental results conclusion is a way to integrate proximity info into ranking in instant-fuzzy search whereas achieving economical time and area complexities. We have a tendency to adapt existing solutions on proximity ranking to instant-fuzzy search. A naive resolution is computing all answers then ranking them, however it cannot meet this high-speed requirement on giant knowledge sets once there are a unit too several answers, so there are a unit studies of early-termination techniques to with efficiency compute relevant answers to beat the area and time limitations of those solutions.

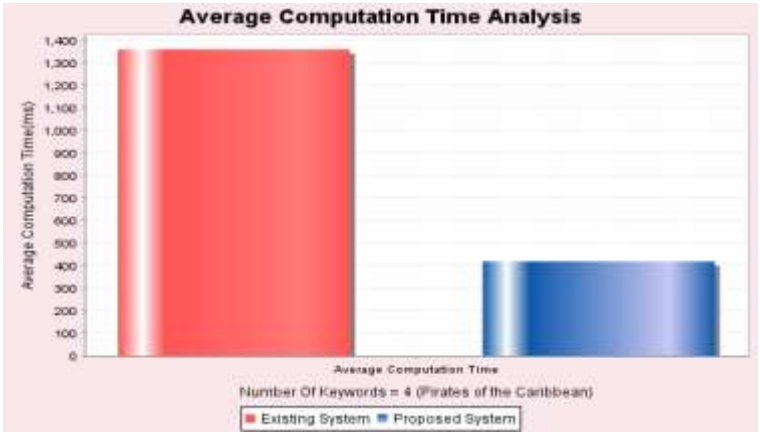

Fig. 1.Average Computation Time analysis for (Pirates of the Caribbean)

Fig. 2 : Average Computation Time Graph vs Number of Records

We have a tendency to propose associate approach that focuses on common phrases within the knowledge and queries, presumptuous records with these phrases area unit graded higher. We have a tendency to study a way to index these phrases associated develop an incremental-computation algorithmic program for with efficiency segmenting a question into phrases and computing relevant answers.

- Find all is good for lengthy keyword search, but it's slightly better for 1-keyword search.
- Query segmentation is good for 2-keyword or 3-keyword search as most of the applications use this and its better as the size of dataset increase.

IV. CONCLUSION

Proximity information is considered to compute top-K answers to improve ranking of instant –fuzzy search system. Existing solutions are used to solve ranking problem such as computing all answers, indexing term pairs and doing early termination. 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 instant fuzzy adaptations of basic approaches. We conducted a very thorough analysis by considering time, space 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.

ACKNOWLEDGMENT

I would like to express my sincere thanks to my guide Asst. Prof. Priti S. Subramanium & H.O.D. Prof. D. D. Patil for his motivation and useful suggestions which truly helped me in improving the quality of this paper and heartily thankful to IJEDR for giving me such a wonderful opportunity for publishing my paper.

REFERENCES

[1]Inci Cetindil,Jamshid Esmaelnezhad, Taewoo Kim and Chen Li, "Efficient Instant-Fuzzy Search with Proximity Ranking", in ICODE, 2014.
[2] A. Nandi and H. V. Jagadish, "Effective phrase prediction," in VLDB, 2007, pp. 219–230.
[3]H. Bast, A. Chitea, F. M. Suchanek, and I. Weber, "Ester: efficient search on text, entities, and relations," in SIGIR, 2007, pp. 671–678.
[4]M. Hadjieleftheriou and C. Li, "Efficient approximate search on string collections," PVLDB, vol. 2, no. 2, pp. 1660–1661, 2009.
[5]S. Ji, G. Li, C. Li, and J. Feng, "Efficient interactive fuzzy keyword search," in WWW, 2009, pp. 371–380.
[6]R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in PODS, 2001.
[7]G. Li, J. Wang, C. Li, and J. Feng, "Supporting efficient top-k queries in type-ahead search," in SIGIR, 2012, pp. 355–364.
[8]M. Persin, J. Zobel, and R. Sacks-Davis, "Filtered document retrieval with frequency-sorted indexes," JASIS, vol. 47, no. 10, pp. 749–764, 1996.
[9]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.
[10]F. Zhang, S. Shi, H. Yan, and J.-R. Wen, "Revisiting globally sorted indexes for efficient document retrieval," in WSDM, 2010, pp. 371–380.