ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015

To Support Search as You type by using MySQL in Databases

Gaikwad Namrata[1], Mahajan Chetan[2], Abhishek Ranjan[3], Manish Kumar[4], Prof. Manisha Galphade[5]

Student, Computer Department, Sinhgad Institute of Technology Lonavala, Pune, India[1,2,3,4]

Assistant Professor, Computer Department, Sinhgad Institute of Technology Lonavala, Pune, India[5]

Abstract: As a user types in a keyword query character by character, the type ahead search or search as you type system computes answers on-the-fly. In this paper we are going to study the way of supporting type ahead search on data residing in databases. We are going to focus on the way to support this type of search using the language, MySQL In this paper we are going to study about the way to use the auxiliary indexes stored on tables so that the search performance can be increased, since we need to achieve an interactive speed which can be done by meeting the high performance requirements of the functionalities. In this paper we present solutions to both single keyword and multikeyword queries. We also allow mismatching between query keyword and answers by developing the novel techniques for fuzzy search by using MySQL. Supporting fuzzy search is important when users have limited knowledge about the exact representation of the entities they are looking for, such as people records in an online directory. Therefor it becomes easy for them to retrieve information with the help of this feature. We provide recent search, history, auto-completion also in this paper.

Keywords: Search As You Type, Type Ahead Search, databases, MySQL, fuzzy search.

I. Introduction

There is tremendous amount of information available on internet almost about every possible thing. It is very easy to retrieve information. The only essential thing is to fire a search query and the information about it will be right in front of you within no time. This is exactly what we are doing in this paper.

Type Ahead Search or Search As you Type helps in retrieving information across records stored in the database as a user types in a query keyword character by character. Most search engines andOnline search forms support auto-completion, which shows the suggested queries or even answers “on the fly”.

In this paper we study how to support search-as-you-type on DBMS systems using the native query language (MySQL). In our paper we implement exact search and fuzzy search.

Exact Search: Exact search is also known as prefix search. As a user types in a query or a partial keyword character by character, the records containing those initial characters or the keywords will be displayed immediately.

Fuzzy Search: Fuzzy Search allows minor mismatches. The system finds the record with keywords that are similar to the query keyword. For example if a user types ‘Motr’ it will show the result as ‘Motorola’. [1]

In this paper we see how to create a Single Page Application (SAP) for Search-As-You-type using the Algorithms namely, Heap algorithm, Scan count Algorithm, Merge skip algorithm and Divide skip Algorithm that are discussed in section 3. The algorithms described, help in supporting multithreading and parallel searching simultaneously.

II. Literature Survey

Our project is influenced from the existing search systems in various DBMS systems like Oracle, Microsoft SQL server, MySQL etc. The existing search systems having some drawback such as - It works only for the specific product, not all of them support the prefix search, and when we use database extender to support prefix search, it is not feasible for databases that do not provide such an extenders for eg. MySQL. When we develop a separate application layer on databases to construct indexes and implement algorithm to search queries, its main drawback is additional hardware costs due to data duplication.
The proposed work is as follows- In our experiment the solution developed on one databases using standard MySQL techniques is portable to other databases. Our goal is to utilize the built-in query engine of the database system as much as possible. In this way, we can reduce the programming efforts to support search-as-you-type. Our MySQL-based techniques enable DBMS systems running on a commodity computer to support search-as-you-type on tables with many records. [1]

III. Algorithms

A. Heap Algorithm

Heap algorithm: The frontiers of the lists as a heap are maintained while merging the lists. At each step, we pop the top from the heap, and increment the count of the record id corresponding to the popped frontier record. This record id is removed from this list, and the next record id on the list (if any) is reinserted to the heap. We report a record id whenever its count is at least the threshold T. Let N = G(Q, q) denote the number of lists corresponding to the grams from the query string, and M denote the total size of these N lists. This algorithm requires O(MlogN) time and O(N) storage space (not including the size of the inverted lists) for storing the heap of the frontiers of the lists. [2]

B. Scan Count Algorithm

The Scan count algorithm improves the Heap algorithm by eliminating the heap data structure and the corresponding operations on the heap. Instead, an array of counts for all the string ids in S is maintained. The N inverted list is scanned one by one. We increment the count corresponding to the string by 1, for each string id on the list. The string ids that appear at least T times on the lists are reported.

```
Input: set of RID lists and a threshold T;
Output: record ids that appear at least T times on the lists.
1.    Initialize the array C of |S| counters to 0's;
2.    Initialize a result set R to be empty;
3.    FOR (each record id r on each given list) {
4.        Increment the value of C[r] by 1;
5.        IF (C[r] == T)
6.        Add r to R;
7.        }
8.        RETURN R;
```

The time complexity of this algorithm is O(M) (compared to O(MlogN) for the Heap algorithm) and the space complexity is 0(S), where lSl is the size of the string collection, since we need to keep a count for each string id. This higher space complexity (compared to O(N) for the Heap algorithm) is not a major concern, since this extra space tends to much smaller than that of the inverted lists.

This algorithm shows that the T-occurrence problem is indeed different from the problem of merging multiple sorted lists into one long sorted list, since we care more about finding those ids with enough occurrences, rather than generating a sorted list. [2]

C. Merge Skip

The main idea of this algorithm is to skip on the lists those record ids that cannot be in the answer to the query, by utilizing the threshold T. Here too just like the Heap algorithm, we maintain a heap for the frontiers of these lists .A key difference is that, during each iteration, we pop those records from the heap that have the same value as the top record t on the heap. Let the number of popped records be n.

If there are at least T such records, we add t to the result set (line 8 in the algorithm), and add their next records on the lists to the heap. Otherwise, we are sure record t cannot be in the answer. In addition to popping these n records, we pop T-1-n additional records from the heap (line 12). Therefore, in this case, we have popped T-1 records from the heap. Let t' be the current top record on the heap. For each of the T-1 popped lists, we locate its smallest record r such that r> t'(line 15).This locating step can be done efficiently using a binary search.

We then push r to the heap (line 16). It is possible to reinsert the same record on the popped lists back to the heap if it is equal to the new top record t'. Those lists that do not have such a record r > t', we do not insert any record from these lists to the heap

```
Input: a set of RID lists and a threshold T;
Output: record ids that appear at least T times on the lists.
1. Insert the frontier records of the lists to a heap H;
2. Initialize a result set R to be empty;
3. WHILE (H is not empty) {
4. Let t be the top record on the heap;
5. Pop from H those records equal to t;
6. Let n be the number of popped records;
7. IF (n >T ) {
8. Add t to R;
9. Push next record (if any) on each popped list to H;
10. }
11. ELSE{
12. Pop T -1 -n smallest records from H;
13. Let t' be the current top record on H;
14. FOR (each of the T -1 popped lists) {
15. Locate its smallest record r > t' (if any);
16. Push this record to H;
17. }
18. }
19. }
20. RETURN R;
```

D. Divide Skip

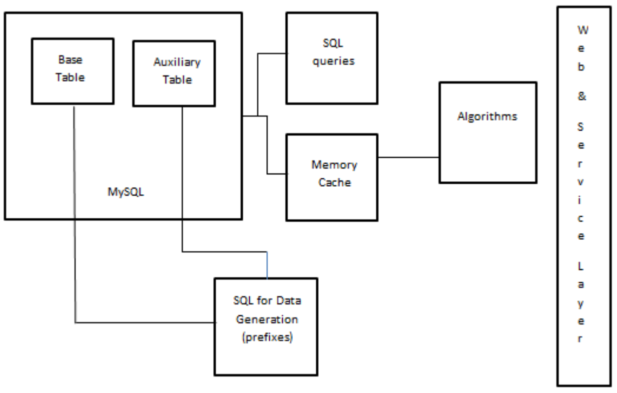
Given a set of RID lists, firstly these lists are sorted based on their lengths. Then they are divided into two groups. The L longest lists are grouped to a set Llong, and the remaining short lists to another set Lshort.

The Merge Skip algorithm is used on Lshort to find records r that appear at least T -L times on the short lists. For every such record r and each list tlong in Llong, we check if r appears on Llong.

This step can be done efficiently using O(log p) time where p is the length of Ilong if the list is implemented as an ordered list, or 0(1) time if the list is implemented as an unordered hash set. If the total number of occurrences of r among all these lists is at least T, then we add it to the result set R.

```
Input: set of RID lists and a threshold T;
Output: record ids that appear at least T times on the lists.
1. Initialize a result set R to be empty;
2. Let Llong be the set of L longest lists among the   lists;
3. Let Lshort be the remaining short lists;
4. Use Merge Skip on Lshort to find ids that appear at least
T – L times
5. FOR (each record r found) {
6. FOR (each list in Ljong)
7. Check if r appears on this list;
8. IF (r appears > T times among all lists)
9. Add r to R;
10. }
11. RETURN R
```

IV. System Architecture



System Architecture of Search-As-You-Type

In this architecture, a MySQL database for storing the data that should be searched is proposed. Since the solution is storing the keywords, reverse key indexes, prefix tables and calculated neighbors are created, the corpus of data used for searching could be from any domain. The data for these auxiliary tables are generated by the use a set of SQL queries on the base tables and some of the other auxiliary tables. In order to speed up execution, some of these jobs could also be taken up in procedural extensions of query languages supported by the respective databases, if desired. Oracle PL/SQL is one such example. We stick mostly to writing SQL queries in order to maintain database neutrality in this paper.

Frequently accessed sets of similar prefixes, as calculated by the algorithms described above in section 3 are cached in the memory using memcache. Most of the implementations would be able to scale well since the set of the data is small. But, if the need arises to cache more and more data as the base tables and the set of keywords on which this feature needs to be supported expands, a distributed implementation of memcache can be taken up. Memcache is also a simple caching solution with simple 'get' and 'put' operations. Its simplicity and high scalability make it a good choice for this implementation. [3]

V. Concusion and Future Work

The proposed project identifies the user search requirements. It evaluates the efficiency of search at time and ease paradigm. It increases the time of search and fakes as if the searching is a lot faster so thus increasing the user satisfaction and providing him right keywords so that user may not loose search result just because simple spelling mistakes or ignorance. In future it is possible to provide centralized profiling for search preferences and also to achieve cross web-app prediction compatibility.

ACKNOWLEDGMENT

We express our sincere thanks to our Guide **Prof. ManishaGalphade**, for her constant encouragement and support throughout our project, especially for the useful suggestions given during the course of project and having laid down the foundation for the success of this work.

We would also like to thank our Project Coordinators, for their assistance, genuine support and guidance from early stages of the project. We would like to thank Prof. Babar, Head of Computer Department for his unwavering support during the entire course of this project work.

References

[1] Guoliang Li, JianhuaFeng, "Supporting Search-As-You-Type Using SQL in Databases", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 25, NO. 2, FEBRUARY 2013
[2] Chen Li, Jiaheng Lu, Yiming Lu, " Efficient Merging and Filtering Algorithms forcApproximate String Searches", Department of Computer Science, University of California, Irvine, CA 92697, USA.
[3] AparnaVedantham, Prof. M. Ganesh Kumar, " ENTERPRISE USE CASES AND IMPLEMENTATION ARCHITECTURE OF SEARCH-AS-YOU-TYPE IN DATABASES USING SQL", INTERNATIONAL JOURNAL OF PROFESSIONAL ENGINEERING STUDIES Volume IV/Issue2/OCT2014.