International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 4 (April 2014), PP.69-73

Survey paper on Generalized Inverted Index for Keyword Search

Dhomse Kanchan[1], Prof. R.L.Paikrao[2]
[1]*M.E. ,Computer, AVCOE, Sangamner*
[2]*H.O.D., Computer Dept., AVCOE, Sangamner*

ABSTRACT:- Inverted lists are normally used to index selected documents to acess the documents according to a set of keywords efficiently. Since inverted lists are normally large in size, many compression techniques have been proposed to minimize the storage space and disk I/O time.This paper presents we propose improved technologies for document identifier assignment a more convenience index structure, the Generalized Inverted Index which merges consecutive IDs in inverted lists into intervals to avoid large storage space. With this index structure, more efficient algorithms can be devised to implement basic keyword search operations. The performance of generalized inverted index is also improved by reordering the documents in datasets using two scalable new algorithms. Experiments on the performance and scalability of generalized inverted index on real datasets prove that Generalized Inverted Index requires minimum storage space as well as increases the keyword search performance, as compared to the old inverted indexes.

Keywords:- keyword search; DocID Assignment, index compression; document reordering, Union Algorithm, Intersection Algorithm.

I. INTRODUCTION

Among the large amount of new information, keyword search is complicated for users to access text datasets. These datasets include textual documents (web pages), XML documents, and relational tables (which can also be regarded as sets of documents). Users use keyword search to retrieve documents by simply typing in keywords as queries. Current keyword search systems normally use an inverted index, a data structure that maps each word in the dataset to a list of IDs of documents in which the word appears to efficiently retrieve documents. The inverted index for a document collection consists of a set of so-called inverted lists, known as posting lists. Each inverted list corresponds to a word, which stores all the IDs of documents where this word appears in ascending order. In practice, real world datasets are so large that keyword search systems normally use various compression techniques to reduce the space cost of storing inverted indexes. Compression of inverted index not only reduces the space cost, but also leads to less disk I/O time during query processing. As a result,compression techniques have been extensively studied in recent years. Since IDs in inverted lists are sorted in ascending order,such as Such as Variable-Byte Encoding (VBE)[1] and PForDelta[2], store the differences between IDs, called d-gaps,and then use various techniques to encode these d-gaps using shorter binary representations. Although a compressed inverted index is smaller than the original index, the system needs to decompress encoded lists during query processing, which leads to extra computational costs. To solve this problem, this paper presents the Generalized Inverted Index which is an extension of the traditional inverted index (denoted by InvIndex), to support keyword search. Generalized Inverted Index encodes consecutive IDs in each inverted list of InvIndex into intervals, and adopts efficient algorithms to support keyword search using these interval lists.

II. BACKGROUND

A. Prior Work on Doc ID Assignment

The compressed size of an inverted list, and thus the entire inverted index, is a function of the d-gaps being compressed, which itself depends on how we assign docIDs to documents (or columns to documents in the matrix). usual integer compression algorithm want fewer bits to represent ta smaller integer than a larger one, but the number of bits required is typically less than linear in the value. This means that if we assign docIDs to documents such that we get many small d-gaps, and a few larger d-gaps, the resulting inverted list will be more compressible than another list with the same average valuebut more uniform gaps. This is the insight that has motivated all the recent work on optimizing docID assignment [3]-[5] .Note that this work is related in large part to the more general topic of sparse matrix compression [6], with parallel lines of work often existing between the two fields.

III. IMPLEMENTATION

A. Basic Concepts of Generalized Inverted Index

An inverted list of file is an index data structure storing a mapping from content, such as digits, to its place in a database file, or in a file or a collection of documents. The main purpose of an inverted index is to permit fast whole text searches, at a cost of increased processing when a document is added to the original database. The inverted data may be the database file itself, rather than its index. It is the most popular data structure used in document storing and accessing the systems,[1] used on a big scale for example in google. Example:

Id	Content
1	Keyword ranking in databases
2	Keyword searching in databases
3	Keyword search in relational databases
4	Required fuzzy type-ahead search
5	Navigation system for any item search
6	Keyword search on relational databases
7	Searching for web databases

Table 1 (a1) Dataset content

(b1) Inverted Index		(c1) Generalized Inverted Index	
Word	**IDs**	**Word**	**Intervals**
Keyword	1,2,3,6	Keyword	[1,3],[6,6]
Databases	1,2,3,6,7	Databases	[1,3],[6,7]

Table 1 (b1),(c1). Example of inverted index for keyword search

B. Search Index Algorithms

Any keyword search method usually helps union and the intersection operations on inverted lists. The union operation is a basic operation to support OR query in which each and every data file that include at least one of the query keywords is returned as an output. The intersection operation is used to support AND query semantics, in which only those data files that include all the query keywords are returned. Usual search algorithms are all based on Identifier lists. This method introduces extra computational costs for decode, and Identifier list based search processes can be very costly because ID lists are usually very large in size.

1) Union operation

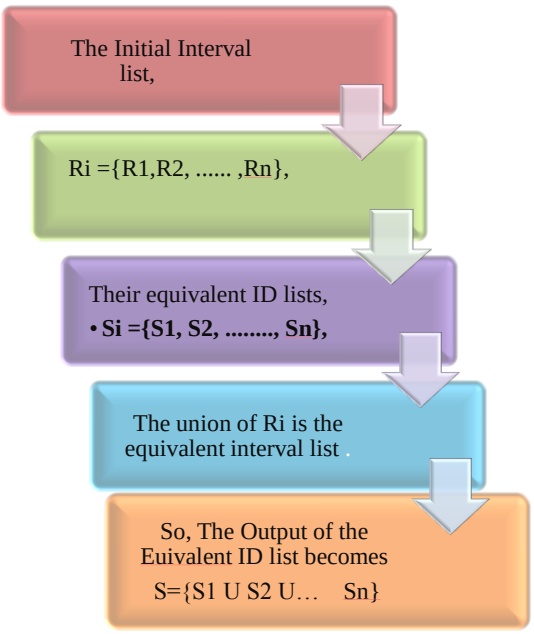

Figure 1: Union Operation

As in set theory, the union of a set of ID lists, denoted by S ={S1, S2, …… , Sn}, is another ID list, in which each ID is contained in at least one ID list in S. Thus the union of a set of interval lists can be defined as follows in figure 1. For example, consider the following three interval lists: {[2, 7], [11, 13]}, {[5, 7], [12, 14]}, and {[1, 3],[.6, 7],[12, 15]. Their equivalent ID lists are { 3, 4, 5, 6, 7, 11, 12, 13}{5, 6, 7, 12, 13, 14}, and {1, 2, 3, 6, 7, 12, 13, 14, 15}. The union of these three ID lists is{1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15}, thus, the union of the three interval lists is the equivalent interval list of this ID list, i.e.{[1, 7],[11, 15]}.

2) Intersection operation

The intersection operation, calculates the intersection list of a set of ordered lists. As with the definitions of the union of interval lists, the intersection of interval lists can be defined as follows in the figure 2.Consider the three interval lists that we have used previously: {[2, 7], [11, 13]}, {[5, 7],[12, 14]}, and {[1, 3], [6, 7], [9, 9], [12, 15]}. Their equivalent ID lists are {2, 3, 4, 5, 6, 7, 11, 12, 13} {5, 12, 13, 14} and, {2, 3, 9, 12, 13, 14, 15}, respectively. The intersection list of these ID lists is{ 12, 13}, thus the intersection of the interval lists is the equivalent interval list of this ID list, i.e., {[12, 13]}.

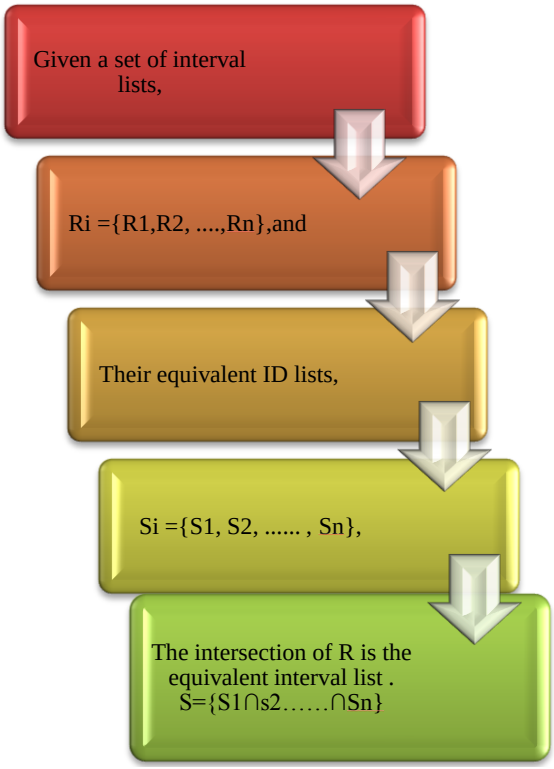

Figure 2: Intersection Operation

3) Twin-heap algorithm

The performance of the basic scan-line algorithm can be increased by maintaining an active interval that shows the interval currently being processed.

Algorithm 1: Twin heap algorithm(R)

I/P: R a set of interval lists
O/P: Z The final interval list
1: Let L be a max heap and U be a min heap
2: for all k ∈[1,n] do
3: let rk be the frontier interval of Rk
4: Insert lb(rk)and ub(rk) to L an U Respectively
5: while U ≠ Φ do
6: Let l be the max element in L
7: Let u be the min element in U
8: if l ≤ u then Add [l,u] to Z
9: Let r∈ Rj be the corresponding interval of u
10: Remove lb(r) from L and pop u from U
11: Let r' be the next interval of r in Rj

12: Insert lb(r') and ub(r') to L and U respectively
13: return Z
However, a single heap is not enough because the lower and upper bounds must be maintained individually.PROBEISECT will run highly because the inverted lists of query keywords usually have very unique lengths.

C. Document reordering concept

Document reordering increases the performance of Generalized inverted index for finding the exact keyword.. This section first explains the necessity of document reordering. Then, since finding the best order of documents is NP-hard, a sorting-based method and a sorting-TSP hybrid method are used to find near optimal solutions.

1) PROBEISECT algorithm

The time complexities of the search algorithms depend on the number of intervals in the interval lists instead of the numbers of IDs .

Algorithm 2: ProbIsect (R)

I/P: R a set of interval lists
O/P: Z The final interval list

1: Sort R in increasing order of list lengths
2: for all r ∈ R1 do
3: R1* ← < r >
4: for all k=2,3,……,n do R*K ← PROBE(r, RK)
5: sum TWINHEAPISECT ({R1* …….. R*n }) to Z
6: return Z
7: procedure PROBE(r, R)
Input: r An interval.
R An interval list.
Output: R* The list of all the intervals in R
that overlap with r.
8: A1← BINARYSEARCH(r.l, R.S)
9: A2← BINARYSEARCH(r.u, R.S)
10: B1← BINARYSEARCH(r.l, R.U)
11: B2← BINARYSEARCH(r.u, R.L)
12: for A ∈ [p1, p2] • do Add [R.Sp, R.Sp] • to R*
13: for B ∈ [q1, q2] • do Add [R.Lq, R.Uq] • to R*
14: Sort R* in ascending order of lower-bounds
15: return R*
16: end procedure

In Generalized Inverted Index contain fewer intervals, the search algorithms will be faster. On the other hand, interval lists containing fewer intervals will require less storage space. Therefore, the search speed and the space cost are both improved by reducing the number of intervals in Generalized Inverted Index.

IV. CONCLUSIONS

This paper describes a generalized inverted index for keyword search in text databases. Generalized Inverted Index has an effective index structure and efficient algorithms to support keyword search. Fast scalable methods enhance the search speed of Generalized Inverted Index by reordering documents in the datasets. Experiments show that Generalized Inverted Index not only requires smaller storage size than the traditional inverted index, but also has a higher keyword search speed. Moreover, Generalized Inverted Index is compatible with existing d gap-based list compression techniques and can improve their performance.

ACKNOWLEDGEMENTS

My sincere thanks go to Amrutvahini College of Engineering for providing me a strong platform to develop my skill and capabilities. This is a nice opportunity for me to present & publish on Generalized Inverted Index for Keyword Search". I am graceful to Prof. Paikrao R.L. for giving me an opportunity to publish this paper.

REFERENCES

[1]. F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel, Compression of inverted indexes for fast query evaluation, in Proc. of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Tammpere, Finland, 2002, pp. 222-229.

[2]. M. Zukowski, S. Hman, N. Nes, and P. A. Boncz, Super-scalar RAM-CPU cache compression, in Proc. of the 22nd International Conference on Data Engineering, Atlanta, Georgia, USA, 2006, pp. 59.

[3]. R. Blanco and A. Barreiro. Characterization of a simple case of the reassignment of document identifiers as a pattern sequencing problem. In Proc. of the 28th annual int. ACM SIGIR conference on Research and development in inf. retrieval, 2005.

[4]. R. Blanco and A. Barreiro. Document identifier reassignment through dimensionality reduction. In Proc. of the 27th European Conf. on Information Retrieval, pages 375–387, 2005.

[5]. R. Blanco and A. Barreiro. Tsp and cluster-based solutions to the reassignment of document identifiers. Inf. Retr., 9(4):499–517, 2006

[6]. D. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In 30th Int. Conf. on Very Large Data Bases (VLDB 2004), August 2004.