Keyword Search Using General Form of Inverted Index

Mrs. Pratiksha P. Nikam
Lecturer
GSMCOE, Pune ,india

Prof. Srinu Dharavath
Professor(Guide)
GSMCOE, Pune ,india

Mr. Kunal Gawande
Software developer
Mumbai , india

Abstract

This Keyword search is the existing method for users to access data from information storage. Inverted lists are usually used to index documents to retrieve documents according to a set of keywords efficiently. Inverted lists are very large and many compression techniques have been proposed to reduce the storage space and disk I/O time. But these techniques perform decompression operation, which increases CPU time. So, Here we present a general form of inverted index which merges consecutive IDs in inverted lists into intervals to save storage space. A more efficient algorithm is devised to do keyword search operations i.e. union and intersection operation by taking advantage of intervals. This algorithm do not require conversions from interval lists back to ID lists, so general form of inverted index is more efficient than traditional inverted index. General form of inverted index improves keyword search performance also it requires less storage space that traditional inverted index.

1.Introduction

Management of string data in databases and information systems has taken on particular importance recently. Recently we study the following problem: given a collection of strings, how to efficiently find those in the collection that are similar to a query string? Keyword search is very difficult for users to access text dataset. If data is very huge, keyword search is critical for users to access text datasets. Datasets may include textual documents (web pages), XML documents, and relational tables. Users use keyword search to retrieve documents by simply typing in keywords as queries. Current keyword search systems usually use an inverted index. Inverted index is 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 called as inverted lists. Each inverted list corresponds to a word, which stores all the IDs of documents where this word appears in ascending order.

Real world dataset is very large so different compression techniques are used to store inverted index. It reduces space cost as well as disk I/O time during query processing. IDs in inverted lists are sorted in ascending order. 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. Our paper address this problem by using general form of inverted index which is an extension of the traditional inverted index to support keyword search. General form of Inverted Index for Keyword Search has become a ubiquitous method for users to access text data in the face of information explosion. Inverted lists are usually used to index underlying documents to retrieve documents according to a set of keywords efficiently. Since inverted lists are usually large, many compression techniques have been proposed to reduce the storage space and disk I/O time. However, these techniques usually perform decompression operations on the fly, which increases the CPU time. This paper presents a more efficient index structure, the General form of Inverted IndeX , which merges consecutive IDs in inverted lists into intervals to save storage space. With this index structure, more efficient algorithms can be devised to perform basic keyword search operations, i.e., the union and the intersection operations, by taking the advantage of intervals. Specifically, these algorithms do not require conversions from interval lists back to ID lists. As a result, keyword search using General form of inverted index can be more efficient than those using traditional inverted indices. scalability of general form of datasets show that it does not only requires less storage space, but also improves the keyword search performance, compared with traditional inverted indexes.

General form of inverted index encodes consecutive IDs in each inverted list of Inverted Index into intervals, and adopts efficient algorithms to support

keyword search using these interval lists. This technique reduces the size of inverted index and support keyword search without list decompression.

This paper covers following points:

- Basics of traditional inverted structure.
- Paper present index structure i.e. General form of index structure . This structure reduces storage space by converting inverted list into interval list.
- Basic operations on interval list such as union and intersection without decompression.

2. Related work and motivation

Most popular data structure for information retrieval is inverted index. For a given collection of document index is defined as follows. Each word in document collection is called term. For each term we maintain a list called inverted list of the entire document in which this word appears. In this list, with each document we may store some score. This score indicate that how much important is this document with respect to that word [6]. Different variants of the inverted index sort the documents in the inverted lists in a different manner. Sorting may be based on document id or score.

Compression techniques are often applied to further reduce space requirement of these lists. 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 remove this extra computation, generalized form of inverted index is developed.

3. General form of inverted index

Many compression techniques have been proposed to reduce the storage space and disk I/O time. However, these techniques usually perform decompression operations on the fly, which increases the CPU time. This paper presents a more efficient index structure, the General form of inverted index, which merges consecutive IDs in inverted lists into intervals to save storage space.

Let $D=\{d_1,d_2\ldots\ldots\ldots d_N\}$ be a collection of documents[2]. Each document in D includes a set of words, and the set of all distinct words in D is denoted by W. Each word $w \in W$ has an inverted list, denoted by I_w, which is an ordered list of IDs of documents that contain the word with all lists sorted in ascending order. List contains ID lists and interval lists. For example, Table 1 shows a collection of 7 document and Table 2 shows its inverted index.

ID	CONTENT
1	Keyword search in databases.
2	Keyword searching using internet browsing in databases
3	Keyword search in multimedia databases.
4	search in multimedia file.
5	Nevigation system for product search.
6	Keyword ranking and search in databases..
7	Searching for hidden web databases.

Table 1. Contents of dataset

Inverted index shown in table 2 corresponds to a word which are four (Searching ,Search ,Keyword ,Databases) most frequent words. AS shown in figure index list is very large as many consecutive IDs for each word. So, we have to compress it.

Word	IDs
Searching	2,7
Search	3,4,5,6
Keyword	1,2,3,6
Databases	1,2,3,6,7

Table 2. Inverted index

Size can be reduced by merging these groups of consecutive IDs into intervals. Each interval r has two numbers ,lower bound(Lb) and upper bound(Ub).This new inverted index are called as general form of inverted index. Table III shows general form of inverted index.

Word	Intervals
Searching	[2,2],[7,7]
Search	[3,6]
Keyword	[1,3],[6,6]
Databases	[1,3],[6,7]

Table 3. General form of inverted index

In above table Search word have [3,6] interval rather than 3,4,5,6 IDs. General form of inverted index is applicable for those datasets whose documents are structured .Relational tables have some attribute values which are shared by many records. And there inverted list contain many consecutive IDs ,as a result general form of inverted index is very small than traditional inverted index. in such datasets, other information in the inverted lists such as the frequency information and position information do not significantly impact either the query processing or result ranking[1]. Our paper considers only structured document and does not consider position information.

If an interval [Lb,U_b] have a single element i.e. Lb =Ub , then two integer are still needed to represent the interval. So for many singe element interval space cost will be more. AS a result we use 3 ID list ie. S for single element interval and Lb ,Ub for lower and upper bounds of multi element intervals. For example <[]2,2],[4,4],[5,5],[6,8],[9,10]> can be written as S = <2,4,5> and Lb =<6,9> and Ub=<8,10>.

Two position indicator x,y are used to indicate the position of S, Lb,Ub.At beginning x,y are set to zero,i.e. they point to first element in S , L_b /Ub. Current interval is found by comparing S_x and Lb_y. if Sx is smaller we return to single element interval[Sx, Sx] and increment x by 1. If Lby is smaller, return the multi-element interval [Lby, Sby] and increment y by 1.

4. KEYWORD SEARCH

Keyword search can be done using union and intersection operations on inverted lists. In union operation the document which contain atleast one of the query keywords is returned as a result .i.e. it support OR query semantic. In intersection operation only those documents that contain all the query keywords are returned. i.e. it support AND semantic.

In traditional keyword search system first retrieves the compressed inverted list for each keyword then decompresses these list into ID lists, and then calculate union or intersection of these list.

4.1 Union Operation

Union operation [2] is denoted by ∪ . Union of a set of ID list denoted by S={ S_1,S_2…….S_n} . S is another ID lists in which each ID is contained in at list one ID list in S .Union of interval list is as follows:
Given a set of interval list R={R_1,R_2…….R_n}and their equivalent ID lists, S={ S_1,S_2…….S_n} ,the union of R is the equivalent interval list of $\cup^{n}_{k=1}$ S_k.
For Example, Consider the following intervals <[1,3],[4,5]> , <[2,6],[13,13]> , <[5,7],[10,11],[13,15]>. Their equivalent ID list are<1,2,3,4,5> , <2,3,4,5,6,13> , <5,6,7,10,11,13,14,15>. Union of these three ID lists is<1,2,3,4,5,6,7,10,11,13,14,15>,thus the union of these interval is equivalent to interval list as<[1,7]>,[10,11],[13,15].

First interval list are converted into ID lists using union operation this method is called as NAIVEUNION algorithm and result again converted back into an interval list.

Scan-Line algorithm is union algorithm without ID-interval conversion. Algorithm use the interval boundaries in the interval lists.. Inspired by the scan-line rendering algorithm in computer graphics[3], The boundaries of all interval in interval lists are first sorted into ascending order, with the scan-line moves from the smallest boundaries to largest boundaries to calculate union list.. The scan line movement maintains a reference counter to count the number of intervals that the scan-line is currently hitting. The counter is decremented by 1 when it hits an upper bound and incremented by 1 when the scan-line hits a lower bound . If counter increases from 0 to 1, it means scan-line is processing an interval an the current boundary is saved in variable a. When the counter decreases from 1 to 0 it means that the scan-line will not hit any interval before it hits another lower-bound .the current boundary is saved in variable b and [a,b] • is returned as the resulting interval. The heap-based merge is used on all the interval lists to enumerate all the lower bounds and upper bounds in ascending order.

Improved scan-line algorithm(SCANLINEUNION algorithm) [2] maintain an active interval to denote current result interval.

Algorithm : SCANLINEUNION (R)
Input: R A set of interval lists.
Output: G The resulting interval list.
1: for all k ε[1, n] • do
2: Let r_k be the first interval of R_k

3: Insert lb(r_k) and ub(r_k) to min-heap H
4: a<-0, b<-0, c<-0
5: while H≠Φ do
6: Let t be the top element in H
7: Pop t from H
8: if t is a lower-bound then
9: c<-C+1
10: if c=1 then α<-t
11: if t is an upper-bound then
12: c<-c-1
13: if c = 0 then b<-t and append [a,b] • to G
14: Let r ε R_j be the corresponding interval of t
15: Let r ' be the next interval (if any) of r in R_j
16: Insert lb(r') and ub(r') to H
17: return G

At the beginning, all pointers are pointing to the first intervals in the interval lists and the active interval is set to be empty. The difference is that only lower bounds are inserted into the heap. In each step, the algorithm first pops up the minimum lower bound in the heap, and then extends the active interval if the two intervals overlap. Finally, the lower bound of the next interval in the corresponding list is pushed into the heap. If the interval corresponding to the popped lower bound (denoted by r) and the active interval do not overlap, active interval is returned as a resulting interval and its lower and upper bounds are updated to lb(r) and ub(r).

4.2 Intersection operation

Intersection operation [2] is denoted by ∩. This operation calculates the intersection list. Intersection of interval list is as follows:

Given a set of interval list R={R_1,R_2…….R_n}and their equivalent ID lists, S={ S_1,S_2……..S_n} ,the intersection of R is the equivalent interval list of $\cap_{k=1}^{n} S_k$.

For Example, Consider the following intervals <[1,3],[4,5]>,<[2,6],[13,13]>,<[4,7],[10,11],[13,15]>. Their equivalent ID list are<1,2,3,4,5>,<2,3,4,5,6,13>,<4,5,6,7,10,11,13,14,15> . The intersection list of these Id list is <4,5>. Thus intersection of interval lists is equivalent interval list of ID list i.e. <[4,5]>. First interval list are converted into ID lists using intersection operation this method is called as NAÏVE intersection algorithm and result again converted back into an interval list. The SCANLINEISECT algorithm enumerates the lower and upper bound in ascending order and returns the intersected intervals based on a reference counter. The performance of the basic scan-line algorithm can be improved by maintaining an active interval that indicates the interval currently being processed. However, a single heap is not sufficient because the lower and upper bounds must be maintained separately. The new TWINHEAPISECT [2] algorithm is illustrated as follows:

Algorithm TWINHEAPISECT (R)
Input : R A set of interval lists.
Output : G The resulting interval list.
1: Let L be a max-heap and U be a min-heap
2: for all k ε[1,n] • do
3: Let rk be the frontier interval of Rk
4: Insert lb(rk) and ub(rk) to L and U respectively
5: while U ≠Φ do
6: Let l be the top (maximum) element in L
7: Let u be the top (minimum) element in U
8: if l <=u then Add [l,u] • to G
9: Let r ε Rj be the corresponding interval of u
10: Remove lb(r) from L and pop u from U
11: Let r0 be the next interval (if any) of r in Rj
12: Insert lb(r')and ub(r') to L and U respectively
13: return G

The TWINHEAPISECT algorithm manages the lower and upper bounds of the frontier intervals in two separate heaps instead of a single heap as in the basic scan-line algorithm. As a result, heap insertions are more efficient than in the basic scan-line algorithm since each heap is 50% smaller (so it takes less time to adjust the heap structures when inserting an element). Thus the TWINHEAPISECT algorithm is more efficient than SCANLINEISECT,

5. RELATED WORK

Users use keyword search to access text data. Keyword search is used for accessing structured or semi structured data, such as XML document and relational databases[4][5]. To answer for keyword queries in keyword search system, inverted indexes are used. Many techniques first convert each ID in an inverted list to difference between it and the preceding ID, called d-gaps, and then encodes the list using compression algorithm. Variable Byte Encoding is used in systems as it simple and provides fast encoding.

6. CONCLUSION

This paper describes the drawback of traditional inverted index and how general form of inverted index overcomes it. Paper describes how to convert inverted list into interval list. General form of inverted index have an effective index structure which require minimum space and an efficient method to provide keyword search.

REFERENCES

[1] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava, *Fast indexes and algorithms for set similarity selection queries*, in Proc. of the 24th InternationaL Conference on Data Engineering, Cancun, Mexico, 2008, pp. 267-276.

[2] Hao Wu_, Guoliang Li, and Lizhu Zhou, *gINIx*, TSINGHUA SCIENCE AND TECHNOLOGY, ISSN 1007-0214 10/12 pp77-87,Volume 18, Number 1, February 2013.

[3] W. J. Bouknight, *A procedure for generation of threedimensional half-toned computer graphics presentations*, Communications of the ACM, vol. 13, no. 9, pp.527-536, September 1970.

[4] G. Li, B. C. Ooi, J. Feng, J.Wang, and L. Zhou, EASE: *An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data*, in Proc. of the ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 2008, pp. 903-914.

[5] G. Li, J. Feng, and L. Zhou, *Interactive search in XML data*, in Proc. of the 18th International Conference on World Wide Web, Madrid, Spain, 2009, pp. 1063-1064.

[6] Manish Patil, Sharma V. Thankachan, Rahul Shah, *Inverted Indexes for Phrases and Strings*, Copyright 2011 ACM 978-1-4503-0757-4/11/07.