NEWS APPLICATION FOR FAST NEWS ALERTS BASED ON ONTOLOGY THEORY AND AUTONOMIC RSS: DISCARDING IRRELEVANT NEWS

Mrs. Shraddha Khonde MS.Ankita Amburle Ms.Prachi Kshirsagar Ms.Samruddhi Bhilare Mrs.Chinmay Deshpande

Abstract—
NEWS APPLICATION FOR FAST NEWS ALERTS BASED ON ONTOLOGY THEORY AND AUTONOMIC RSS: DISCARDING IRRELEVANT NEWS is an application that accepts and processes requests from the patron: end users. Besides the local server database (for storing keywords).This application also integrates databases from online news and newspapers. To maintain the speed of the news retrieval we aim at building a parser to parse the RSS of various international news papers. The search engine such as Google, Bing, yahoo enables users to express search query by means of one or more keywords. This paper proposes a system called generalized inverted list for keyword search. The main function of inverted lists is to enable fast full text searches at increased speed when a document is submitted to database. Since inverted lists are large, some techniques are projected to reduce storage space and disk I/O time. However, we propose more efficient index structure called GINIX (Generalized inverted index) that groups consecutive ID's in inverted list into intervals to save storage space. The system performance can be increased by using two scalable algorithms. The evaluation results shows that GINIX requires less space and improves the keyword search performance compared with existing inverted indexes.

Keywords: keyword search; index compression; document reordering

---

1. INTRODUCTION

Keyword search is a crucial technology. Search engines that index the Web provide a spread and ease of access to knowledge that was impossible only a decade ago. Keyword search has also grown a significant at the other end of the size spectrum. For example, the services built into web rely on active text search, Queries are evaluated by processing the index to identify matches which are then returned to the user. However, there are also many differences. Database systems must contend with arbitrarily complex queries, whereas the vast majority of queries to search engines are lists of terms and phrases. Queries are evaluated by processing the index to identify matches which are then returned to the user. However, there are also many differences. Database systems must contend with arbitrarily complex queries, whereas the vast majority of queries to search engines are lists of terms and phrases.

1.1Application

The application will fetch all the current information from the sources like Newspapers, Blogs, and the related online available source etc. and therefore it includes three phases:

- Input query

The user will input query to the search engine which he/she want to retrieve information. For ex
Input query='Business'

- Search results

The search engine will investigate all the recent information about the keyword 'Business' using GINIX.

- Display Results

and desktop search systems help users locate the links and information on the web. Search engines are structurally similar to database systems. Documents are stored in a warehouse, and an index is maintained.

The related information about 'Business' will display according to the index.
The following system architecture gives you the brief idea about how the application works.

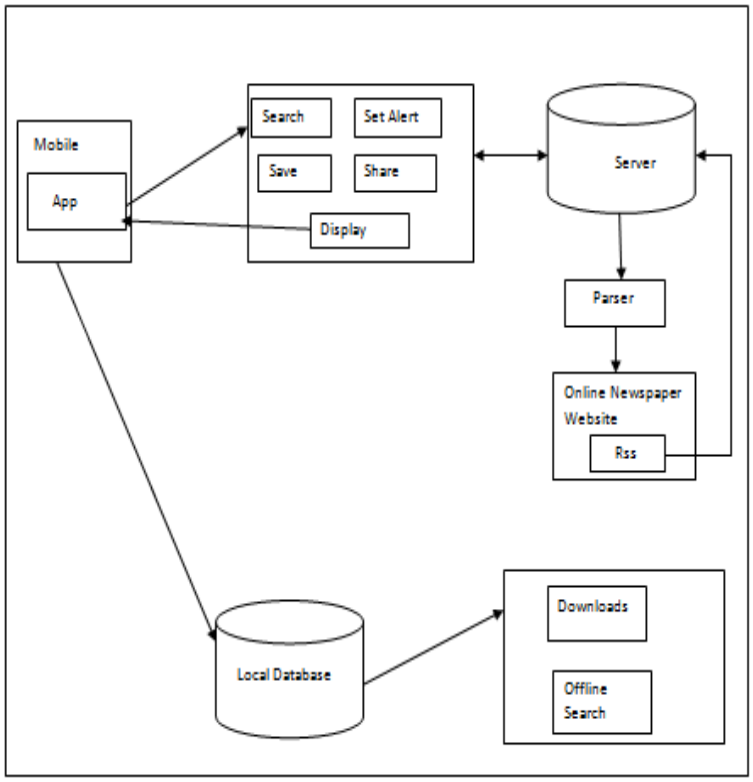

FIG 1.SYSTEM ARCHITECTURE

2. Related Work

The authors have performed several works before. However, 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. The technique of inverted indexes for fast query evaluation is proposed by [1]. W. Sheih presented a document identifier assignment technique for inverted file index compression[2]. H.Yan proposes a Inverted index compression and query processing with optimized document ordering[3]. A.Chandel presents an algorithm for fast indexes and selected queries[4]. H. Wang shows a technique of ranked keyword search on graphs known as BLINKS. It is a bi-level indexing and query processing scheme for top-k keyword search on graphs based on backward search strategy[5]. J. Zobel proposed an system that uses inverted file for text search engine. Search Queries are evaluated by processing the index to identify matches which are then returned to the user[6]. G.Li performs the analysis on An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. It is a novel ranking mechanism for enhancing search effectiveness[7] V.N.Anh proposes a new approach allows extremely fast decoding of inverted lists during query processing, while providing compression rates better than other high-throughput representations[8].J.Zhang includes caching mechanism in inverted index that search engines use several levels of caching to improve query throughput[9].

3. Proposed Work

This paper proposes three types of algorithm:

3.1 Scan line algorithm

The scan-line movement maintains a reference counter to count the number of intervals that the scan-line is currently hitting. The counter is incremented by 1 when the scan-line hits a lower bound and is decremented by 1 when it hits an upper bound. If the counter increases from 0 to 1 (which means that the scan-line. is processing an interval), the current boundary is saved in variable a. When the counter decreases from 1 to 0 (which 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.

Input: X → set of interval lists
Output: Z→ resulting interval list

1. for all k ϵ [1,n] do
2. Let x1 be the first interval X_m
3. Insert lb(x1) and ub(x2) to min heap H
4. a←0,b←0,c←0
5. while H≠ ϕ
6. Let t be the top element in H
7. Pop t from H
8. if t is lower bound then
9. c←c+1
10. if c=1 then a←1
11. if t is upper bound then
12. c←c-1
13. if c=0 then b←1 and then append[a, b] to G
14. Let r ϵ R_j be the corresponding of t
15. Let r′ be the next interval or r in R_j
16. Insert lb(r′) and ub (r′) to H
    return Z

3.2 Improved Scan line algorithm

The performance of the scan-line-based algorithm can be improved by maintaining an active interval to denote the current result interval. Similar to the SCANLINEUNION algorithm, 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/. The details of this algorithm, called the SCANLINEUNION+ algorithm.

Input: X → set of interval lists
Output: Z→ resulting interval list

1 for all k ϵ [1,n] do

2. Let x1 be the first interval X_m
3. Insert lb(x1) to min heap H
4. a←0,b←0,
5. while H≠ ϕ

6. Let t be the top element in H
7. Let X ϵ Rj be the corresponding interval of l
8. if b<1 and a≤b then Add [a,b] to G
9. else
10. a←1
11. if b<ub(r) then b ←ub(r)
12. pop l from H
13. Let r′ be the next interval or r in R_j
14. Insert lb(r′) to H
15. if a≤b then Add [a,b] to G
return Z

3.3 Twin Heap algorithm

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 algorithm is illustrated in 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 TWIN HEAPISECT algorithm is more efficient than SCANLINEISECT.

Input: X → set of interval lists
Output: Z→ resulting interval list
1.
2. Let X be the min heap and Y be the max heap
3. for all k ϵ [1,n] do
4. Let q be the frontier interval of Rk
5. Insert lb(r_k) and ub(r_k) to min heap H
6. Let m be the top(max) element in X
7. Let n be the top(minimum) element in Y
8. if m≤n ,add [m,n] to G
9. Let r ϵ R_j be the corresponding interval of n
10. Remove lb(r) from X and pop n from Y
11. Let r′ be the next interval or r in R_j
12. Let lb(r′) and ub (r′) to X and Y respectively.
return Z

3.4 Probe-based algorithm

The probe-based intersection algorithm usually runs faster for ID lists than the merge-based intersection algorithm in real applications. A similar idea is used here to devise a probe-based algorithm to accelerate the interval list intersection process. Specifically, each interval in the shortest interval list is enumerated while the other interval lists are probed for intervals that overlap with it.

Input: X → set of interval lists
Output: Z→ resulting interval list

1. Sort x in ascending order of list lengths
2. for all x ϵ X1 do
3. X1* ← (x)
4. for k=2,3,……..n do X_k* ←PROBE(x,X_k)
5. Add TWIN HEAP ISECT ({X1*…….Xn*}) to Z
6. return Z
7. procedure probe(x,X)
   Input: x: an interval
   X: an interval list

Output: X*: The list of all the intervals in X that overlap with x

p1←Binary search(x,l,X,S)
p2←Binary search(x,u,X,S)
q1←Binary search(x,l,X,U)
q2←Binary search(x,u,X,L)
for pϵ[p1,p2] do Add [X.Sp,X.Sp] to X*
sort X* in ascending order of lower bounds
return X*

end procedure

4. Conclusion

We proposed a system called GINIX(Generalizes inverted index) for keyword search in text database has an effective index structure and efficient algorithms to support keyword search. Fast scalable methods enhance the search speed of Ginix by reordering documents in the datasets. The evaluation result shows that Ginix not only requires smaller storage size than existing inverted index, but also consists of greater keyword search speed.

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] W. Shieh, T. Chen, J. J. Shann, and C. Chung, Inverted file compression through document identifier reassignment,

Information Processing and Management, vol. 39, no. 1, pp. 117-131, 2003
Management of Data (SIGMOD '08), pp. 1087-1098, 2008.

[3] H. Yan, S. Ding, and T. Suel, Inverted index compression and query processing with optimized document ordering, in Proc. of the 18th International Conference on World Wide Web, Madrid, Spain, 2009, pp. 401-410.

[4] 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.

[5] H. He, H. Wang, J. Yang, and P. S. Yu, BLINKS: ranked keyword searches on graphs, in Proc. of the ACM SIGMOD International Conference on Management of Data, Beijing, China, 2007, pp. 305-316.

[6] J. Zobel and A. Moffat, Inverted files for text search engines, ACM Computing Surveys, vol. 38, no. 2, pp. 6, 2006

[7] 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.

[8] V. N. Anh and A. Moffat, Inverted index compression using word-aligned binary codes, Information Retrieval, vol. 8, no. 1, pp. 151-166, 2005.

[9] J. Zhang, X. Long, and T. Suel, Performance of compressed inverted list caching in search engines, in Proc. of the 17th International Conference on World Wide Web, Beijing, China, 2008, pp. 387-396.

[10] S. Agrawal, S. Chaudhuri, and G. Das, DBXplorer: A system for keyword-based search over relational databases, in Proc. of the 18th International Conference on Data Engineering, San Jose, California, USA, 2002, pp. 5-16.