Index Tuning for Query-log based On-line Index Maintenance

Sairam Gurajada[*]
Max-Planck-Institut für Informatik
Saarbrücken, Germany
gurajada@mpi-inf.mpg.de

Sreenivasa Kumar P.
Indian Institute of Technology Madras
Chennai, India
psk@cse.iitm.ac.in

ABSTRACT

The existing query-log based on-line index maintenance approaches rely on frequency distribution of terms in the static query-log. Though these approaches are proved to be efficient, but in real world, the frequency distribution of the terms changes over a period of time. This negatively affects the efficiency of the static query-log based approaches. To overcome this problem, we propose an index tuning strategy for reorganizing the indexes according to the latest frequency distribution of the terms captured from query-logs. Experimental results show that the proposed tuning strategy improves the performance of static query-log based approaches.

Categories and Subject Descriptors: H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—*Indexing Methods*; H.3.2 [Information Storage and Retrieval]: Information Storage—*File Organization*; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*Search Process*

General Terms: Algorithms, Performance

Keywords: Inverted File, Inverted Index, Search Engine, Query Log

1. INTRODUCTION

Inverted indexes are an important and widely used data structure for index maintenance in Information Retrieval (IR) systems. They comprise of *dictionary* (for storing terms) and *postings lists* (for storing postings of individual terms) [12]. Postings represent the information about a document for a particular term. Based on the requirement, the postings store information as little as document ID to as large as positional information of term in documents, term frequencies etc. A query to an IR system is a list of terms along with constraints. The result is a list of documents obtained by applying the constraints in the query, which is then ordered by a ranking method.

Based on the temporal characteristics of the document collections, an inverted index is built either in off-line mode or on-line mode. In off-line mode, it is assumed that the entire document collection is known and remains static during index construction. The final index is a single large inverted index. Though we get high query performance, this approach is not applicable to the dynamic corpus of documents. On-line indexing on the other hand is designed for such dynamic scenarios. They provide flexibility of querying the index and alter the document collection during index construction. This type of indexing is needed for the cases where queries should be processed readily after a document is indexed, eg., news search.

Many algorithms [1, 11, 5, 10, 8, 2, 3, 4, 6] were proposed in the recent past to perform indexing dynamic document collections. Recently, Gurajada et al. [7] proposed a new selective merge-based on-line index maintenance approach based on frequency distribution of query-logs. They maintain separate indexes for frequent and infrequent terms and adopt different merge strategies. Terms are classified into frequent and infrequent a priori using a static query-log. The use of static query-logs has performance limits when the frequency distribution of existing terms is altered. In addition, any new terms appearing in the new logs are treated as infrequent and do not benefit from the index organization. For example, the query *Barrack Obama* which was rarely appearing before 2008 has later became highly frequent. The query, for instance *Twitter*, which is not present few years back is now frequently queried. Such cases are inefficiently handled by static query-log based approaches. To handle this problem, we propose an index tuning strategy for these approaches to address dynamic query-logs.

The central idea of the index tuning is to consider frequency distribution of new query-log and identify the misplaced terms. The misplaced terms (called as *diff-terms*) along with their postings list form a *diff-term index*. The diff-term index is then handled like an other auxiliary index and added to the existing index by query-log based approaches.

The contributions of this paper are:

- We present an *index tuning* strategy to improve the index maintenance performance for dynamic query-logs.
- We provide a performance study of our strategy applied on existing query-log based on-line indexing approaches.

[*]The work was carried out when the author was at Indian Institute of Technology Madras, India

2. BACKGROUND

Here, we discuss a brief overview of the query-log based approaches on which index tuning is applied, and the AOL query-log analysis which forms the basis for our proposed index tuning strategy.

2.1 Query-log based on-line indexing

In [7], authors proposed a merge based on-line indexing approach using the frequency distribution of terms in the query-log. The pivotal idea of query-log based approaches is to partition the index into two: frequent-term index and infrequent-term index, and adopt a different merge policy for each. For partitioning the index, they proposed a *horizontal partitioning* technique in addition to the vertical partitioning approach. Horizontal partitioning splits the index using a *split-point* (threshold) calculated by applying *pareto* principle (80-20) on frequency distribution of terms in the query-log. Frequent terms (>*split-point*) along with their postings lists form frequent-term index and the rest form infrequent-term index.

Based on the horizontal partitioning, three query-log based approaches are proposed in [7] - Single-split Immediate Merge, Single-split Multi-partition, and Multi-split Indexing approach. These three approaches follow a general paradigm used for on-line indexing. An auxiliary index is maintained in the main memory and continuously updated with new documents. Once the auxiliary index size reaches the allowed quota, it is either merged with an on-disk index (with cascading merges) or moved as a new partition. Query-log based approaches are multi-partition approaches, i.e. they result in more than one on-disk index to exist simultaneously. These approaches generalize the idea of Logarithmic merge and Geometric partitioning approaches and use horizontal partitioning for splitting the indexes. A brief overview of the query-log based approaches is given below:

i. Single-split Immediate Merge

In Single-split Immediate Merge, the auxiliary index, when it is required to be moved to disk, is merged with the index at Partition 0. Based on user defined value of r (maximum no. of merges allowed), the index at Partition 0 is split into frequent-term and infrequent-term index using horizontal partitioning. The frequent-term index is maintained using immediate merge policy and infrequent-term index by a lazy merge policy (a generalized logarithmic merge). This indexing scheme provides a high degree of query performance since frequent-term index is a large single partitioned index. However, the index maintenance costs are expensive but better than naïve immediate merge approach.

ii. Single-split Multi-Partition

In Single-split Immediate Merge, the advantages of using lazy merge policy for the large infrequent-term index are shadowed by the use of very expensive immediate merge policy for frequent-term index. This limits the Single-split immediate merge policy from providing better trade-off between index maintenance costs and query performance. Single-split Multi-partition overcomes this problem by adopting an incremental multi-partition active merge policy (a generalized geometric partitioning) for maintaining frequent-term index. This indexing approach achieves a much better trade-off than the one offered by Single-split immediate merge.

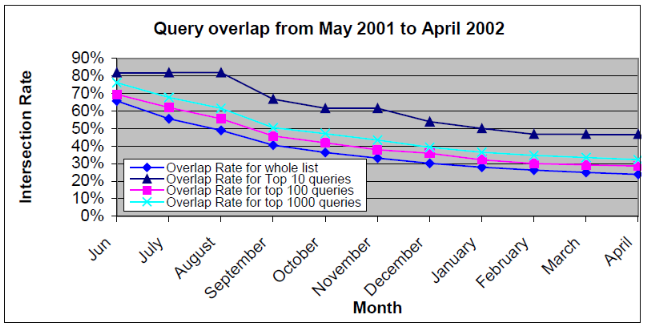

Figure 1: Graph representing overlap rate of AOL query-log from May 2001 to April 2002[1]

iii. Multi-split Indexing

Although Single-split multi-partition provides a better trade-off and performs better than Geometric partitioning, there are some short comings with this approach. For instance, terms those fall below the split-point are treated identically to the terms that rarely appear in query-log. Similarly, terms that just crossed the split-point enjoy an expensive merge policy. This impacts the performance of single-split multi-partition both in index maintenance via first instance and query performance through second instance. To overcome this, multi-split indexing leverages the power law distribution of term frequencies and recursively defines multiple split-points. Multi-split on-disk indexes are the result of applying recursive horizontal partitioning better represented by an *index tree*. This approach achieves a better trade-off than Single-split multi-partition approach.

2.2 AOL query-log analysis

G Pass et al. [9] performed analysis over AOL query-log and presented some temporal characteristics of queries. They used two measures *Overlap rate* and *Correlation Coefficient* to analyse the query-logs from May 2001 to April 2002. Their findings[1] in Figure 1 shows that the percentage overlap of queries in two different query-logs decreases with time. The overlap rate on average for consecutive months is 0.905, consecutive quarters is 0.833, consecutive semi-yearly is 0.795. For the whole year only $20 - 30$ % of all unique queries survive, and the longer a query survives the more likely it survives in future logs. This suggests that the use of static query-log for index maintenance is inefficient and index tuning is essential to make query-log based approaches efficient.

3. INDEX TUNING

Index Tuning refers to transformation of the existing index into a new index. The new index incorporates the changes to the term frequencies observed in the new query-log. Index tuning is a three step process: 1) Identifying the *diff-terms*, terms whose frequency is changed, 2) Building a *diff-terms index* for diff-terms, and 3) Merging the diff-terms index with the existing on-disk index.

Identifying the diff-terms: The first step in the index tuning process is to identify diff-terms. We define a *diff-log*, which comprises of the terms whose frequencies had changed

[1]Query-Log Analysis, www.ir.iit.edu/∼abdur/publications /QueryResearch.pdf

Algorithm 1: IndexTuning(onDiskIndexroot, diffLog)

```
begin
    for each term in diffLog do
        pList ← extractPostings(onDiskIndexroot,term);
        addList(diffTermIndex,pList);
    InsBlock(onDiskIndexroot, diffTermIndex) ;
end
```

from old query-log to the new query-log. Since over a time span, a large number of terms have frequency changes, it is inefficient to consider all of them to be recorded into diff-log for index tuning. Instead, we record only the terms which are mis-classified according to new log, i.e we take the new query-log and compute the new thresholds based on 80-20 principle. Mis-classified terms (or diff-terms) are the terms which are infrequent in old query-log and frequent in new query-log or vice-versa.

Building diff-term index: After the diff-terms are identified, the next step is to build a diff-term index. During the diff-term index construction, the postings list of the terms are extracted from the existing on-disk index and grouped together to form the diff-term index. Extracting the postings list for a term is similar to querying the term, this involves traversing through all the indexes for collecting its postings. Every time a postings list for a term is extracted from an index, the postings are not deleted immediately, but they are copied and marked as delete for efficiency purpose. The final postings list for a term is the concatenation of all the identified postings. This list is added to the diff-term index.

Merging the diff-terms index with on-disk index: The final step involved is to merge the diff-term index with the existing on-disk index. The merging step is similar to the merging of an auxiliary index with the on-disk index. The diff-term index is first merged with the on-disk index at partition 0, which can trigger cascading merges. Before invoking the merging process, the split-points (threshold) are updated to the new split-points that are obtained from the new query-log using the 80-20 % split principle.

Algorithm 1 describes the general index tuning approach. For each term in the computed diff-log, first its postings are extracted from the on-disk index and the term along with its postings are added to the diff-term index. After processing all the terms in diff-log, the final diff-term index is merged with existing on-disk index (by the InsBlock() described in [7]). Since the three query-log based approaches follow different merging schemes, they result in different costs for index tuning mainly influenced by the extraction process costs.

Extraction in Single-split approaches

Algorithm 2 describes the process of extracting the postings for a given term `term`. First the `term`'s frequency is obtained from old query-log and compared with the split-point. If the `term` is frequent, all the frequent-term indexes are queried, otherwise infrequent-term indexes are queried for extracting postings. For efficiency reasons, extracting postings list operation does not delete the postings but marks them as deleted in the partitions which contain the term. Subsequently during the merge phase, the postings are deleted in those partitions which participate in merging.

Algorithm 2: extractPostings(onDiskIndexroot, `term`)

```
begin
    // For Single-split approaches;
    node ← onDiskIndexroot;
    if frequent(term) then
        // For SSIM, sizeOf(frequentIndexesList) is 1;
        for each index in frequentIndexesList do
            pList ← extract(frequentIndex, term);
    else
        for each index in inFrequentIndexesList do
            pList ← extract(index, term);
    return pList;
end
```

Algorithm 3: extractPostings(onDiskIndexroot, `term`)

```
begin
    // For Multi-split approach;
    node ← onDiskIndexroot;
    if node is NULL then
        return NULL;
    pList ← extract(node,term);
    if frequent(term) > nodeThreshold then
        pList ← concat(pList,extractPostings(node→right);
    else
        pList ← concat(pList,extractPostings(node→left);
end
return pList;
```

Extraction in Multi-split approaches

Multi-partition approaches maintain the indexes that are represented by an index tree. Extraction, similar to querying, is a recursive binary search over the index tree. After extracting the postings list at a node, if the term's frequency is greater than node's split-point value the extraction continues over right sub-tree otherwise left sub-tree partitions are considered. Since the average query times for multi-split approaches are slightly expensive compared to Single-split multi-partition approaches, the index tuning costs also follow the same pattern. The algorithm for extracting postings list for a term is presented in Algorithm 3.

4. PERFORMANCE STUDY

Datasets: For our performance study of index tuning strategy applied over the three query-log based on-line indexing approaches, we used Wikipedia[2] collection of 85GB containing 7 million documents. The average size of the document in the collection is around 12KB. For evaluating the query performance, we used AOL query-log containing 1.2 million terms. For the performance study of index tuning approaches, we generated a synthetic query-log from AOL query-log. The synthetic query-log is generated by taking all the queries from the original query-log and changing their frequency of occurrence. For our test purpose, we made 20% of random queries change from frequent to infrequent or vice-versa. From the synthetic log, we computed the diff-log, that contains only the terms which are frequent in old query-log and infrequent in new query-log or the terms which are infrequent in old query-log and frequent in new log.

We performed the evaluation of index tuning over three on line indexing approaches: 1) Single-split Immediate merge

[2]http://static.wikipedia.org

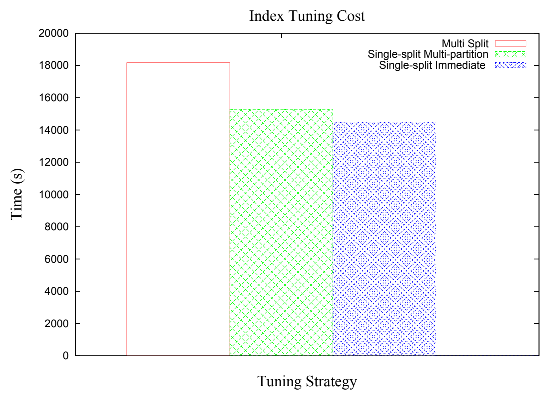

Figure 2: Index tuning costs for three on-line indexing approaches

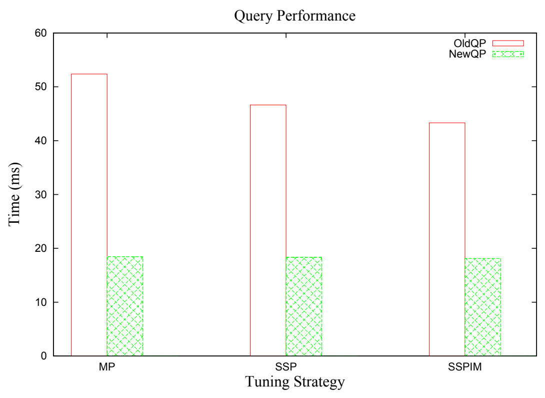

Figure 3: Query performance after index tuning

2) Single-split Multiple Partition approach and 3) Multi-partition approach. In the performance study, we evaluated the time taken for tuning the existing on-line index into a new one which is based on the new query-log, and also compared the average time taken for each query in old index and new index.

Figure 2 shows the evaluation of the index tuning costs for the three approaches. Here, one can observe that the index tuning costs for Single-split Immediate merge index is lower compared to the other approaches. This is because, the extraction of postings from an index for a term is proportional to the querying time taken for that term. For our test case, we observed that index tuning cost for Single-split Immediate Merge is **7%** less of the Single-split Multiple Partition approach and **25%** less of the Multi-split Indexing approach.

The evaluation query times after the index tuning for the three approaches is shown in Figure 3. Here, we can observe that the new query times are almost same for all the three approaches. This is because, we have taken only the diff-log queries for evaluation. Since all the diff-log terms form a single index after index tuning and merged with the existing on-disk index, we have the same query time for diff-terms with respect to all the approaches. But this will change once the usage of on-line indexing is resumed. From our experimental study, we observed a significant improvement in query performance. For Single-split Immediate merge (SSIM) the query performance is improved by **58%**, for Single-split Multiple Partition approach (SSMP) the improvement is by **61%**, and for Multi-split approach the improvement is by **64%**.

5. CONCLUSIONS

In this paper, we propose an index tuning approach for query-log based on-line index strategies. The proposed approach overcomes the problems faced by query-log based approaches with dynamic query-logs. We exploit the frequency distribution of query-logs to compare and identify the terms that are inefficiently handled in the existing index and use a tuning approach to reorganize the existing index. Index tuning cost for a query-log based approach is directly proportional to its query costs. As index tuning reorganizes the existing index, it substantially improves the query performance for the terms which are inefficiently handled before by query-log based approaches. Thus, the use of index tuning helps in improving the performance of query-log based approaches and make them viable to dynamic changes in query-logs.

6. ACKNOWLEDGEMENTS

We would like to thank Max-Planck-Institut für Informatik, Saarbrücken, Germany for their generous support.

7. REFERENCES

[1] E. W. Brown, J. P. Callan, and W. B. Croft. Fast incremental indexing for full-text information retrieval. In *VLDB '94*, pages 192–202, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] S. Büttcher and C. L. A. Clarke. Indexing time vs. query time: trade-offs in dynamic information retrieval systems. In *CIKM '05: Proceedings of the 14th ACM CIKM*, pages 317–318, New York, NY, USA, 2005. ACM.

[3] S. Büttcher, C. L. A. Clarke, and B. Lushman. Hybrid index maintenance for growing text collections. In *In SIGIR 2006: Proceedings of the 29th annual international ACM SIGIR*, pages 356–363, 2006.

[4] T. Chiueh and L. Huang. Efficient real-time index updates in text retrieval systems. Technical report, Experimental Computer Systems Lab, Department of Computer Science, State University of New, 1998.

[5] D. Cutting and J. Pedersen. Optimization for dynamic inverted index maintenance. In *SIGIR '90: Proceedings of the 13th annual international ACM SIGIR*, pages 405–411, New York, NY, USA, 1990. ACM.

[6] R. Guo, X. Cheng, H. Xu, and B. Wang. Efficient on-line index maintenance for dynamic text collections by using dynamic balancing tree. In *CIKM '07: Proceedings of the sixteenth ACM CIKM*, pages 751–760, New York, NY, USA, 2007. ACM.

[7] S. Gurajada and S. K. P. On-line index maintenance using horizontal partitioning. In *Proceeding of the 18th ACM CIKM*, CIKM '09, pages 435–444, New York, NY, USA, 2009. ACM.

[8] N. Lester, A. Moffat, and J. Zobel. Fast on-line index construction by geometric partitioning. In *CIKM '05: Proceedings of the 14th ACM CIKM*, pages 776–783, New York, NY, USA, 2005. ACM.

[9] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 1, New York, NY, USA, 2006. ACM.

[10] I. M. Strategies, N. Lester, J. Zobel, and H. E. Williams. In-place versus re-build versus re-merge:. In *In Proceedings of the 27th Conference on ACS*, pages 15–23. Society, Inc, 2004.

[11] A. Tomasic, H. García-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD*, pages 289–300, New York, NY, USA, 1994. ACM.

[12] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes (2nd ed.): compressing and indexing documents and images.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.