Hybrid Index Maintenance for Growing Text Collections

Stefan Büttcher Charles L. A. Clarke Brad Lushman

School of Computer Science
University of Waterloo, Canada

{sbuettch,claclark,bmlushma}@plg.uwaterloo.ca

ABSTRACT

We present a new family of hybrid index maintenance strategies to be used in on-line index construction for monotonically growing text collections. These new strategies improve upon recent results for hybrid index maintenance in dynamic text retrieval systems. Like previous techniques, our new method distinguishes between short and long posting lists: While short lists are maintained using a merge strategy, long lists are kept separate and are updated in-place. This way, costly relocations of long posting lists are avoided.

We discuss the shortcomings of previous hybrid methods and give an experimental evaluation of the new technique, showing that its index maintenance performance is superior to that of the earlier methods, especially when the amount of main memory available to the indexing system is small. We also present a complexity analysis which proves that, under a Zipfian term distribution, the asymptotical number of disk accesses performed by the best hybrid maintenance strategy is linear in the size of the text collection, implying the asymptotical optimality of the proposed strategy.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Textual databases; H.3.4 [**Systems and Software**]: Performance evaluation

General Terms

Experimentation, Performance

Keywords

Information Retrieval, Index Construction, Index Maintenance, Merge, In-Place, Hybrid

1. INTRODUCTION

Index maintenance strategies for text retrieval systems in dynamic search environments, where index update operations are interleaved with search queries, have been studied intensively over the past few years. In general, every maintenance strategy can be described as a trade-off between index maintenance performance and query processing performance. Index update operations can be carried out very efficiently if we do not care about query processing performance; spending more time on index reorganization tasks, on the other hand, usually results in lower response times when processing search queries.

Although fully dynamic text collections allow document insertions, deletions, and modifications, in this paper we only focus on document insertions and discuss the problem of maintaining an inverted index for a monotonically growing text collection — a restriction that is quite common in this area [6] [7] [11] [12] [14]. For a discussion of index maintenance strategies in the presence of document deletions, see Chiueh and Huang [3] or Büttcher and Clarke [1].

Virtually all existing index maintenance strategies for monotonically growing text collections work by accumulating postings for incoming documents in main memory, building an in-memory inverted file, and only combining this in-memory index with the existing on-disk data structures when a certain, pre-defined memory utilization threshold is exceeded. This way, disk accesses can be amortized over a larger number of index update operations, resulting in increased indexing performance. The individual update strategies only differ in how they combine the accumulated in-memory data with the data stored on disk.

Traditionally, index maintenance strategies for text retrieval systems based on inverted files have either employed an in-place [14] or a merge-based [6] update scheme. In an in-place update scheme, whenever the data in memory have to be combined with the existing on-disk index, the in-memory posting lists are appended to the existing ones on disk. In general, this requires relocating existing on-disk lists. These relocations are time-consuming and can be avoided by using an overallocation strategy that leaves some amount of free space after every on-disk posting list. Only when the amount of free space is too small for the postings that are currently held in memory, the whole posting list has to be moved to a new location, where there is enough space for all postings for the given term. In contrast, merge-based strategies create a new on-disk inverted file by combining an existing one with the in-memory data. Although this requires a great number of disk operations, since the entire on-disk index has to be read, it usually can be done fairly efficiently because the number of disk seeks involved is very small, thanks to the largely sequential disk access pattern.

Roughly speaking, merge-based strategies are better at dealing with short posting lists, while in-place methods are better at handling long posting lists. Recently, we have pre-

SIGIR'06, August 6–10, 2006, Seattle, Washington, USA.

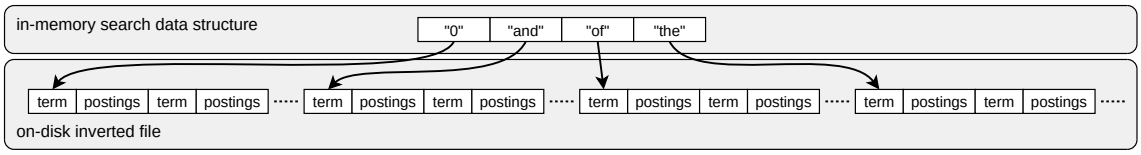

Figure 1: General structure of an inverted file (schematical). Usually, a B-tree or a sorted list is chosen as the in-memory data structure. If the terms and their posting lists are stored in lexicographical order in the on-disk index, only a very small number of term descriptors needs to be kept in memory.

sented a family of hybrid strategies, based on a distinction between short and long posting lists [2]. Hybrid strategies maintain short posting lists following a merge-based approach, while long lists are updated in-place. By combining the two paradigms, they achieve better indexing performance than either approach alone.

In this paper, we present a new family of hybrid index maintenance strategies. Our new approach is based upon the same idea to distinguish between short and long posting lists and to treat them in different ways, but it corrects some shortcomings that we found in our previous technique. The new approach is similar to the pulsing technique proposed by Cutting and Pedersen [4] and allows the indexing system to view a posting list as the concatenation of a long list and a short list. It offers better indexing performance than the old one, especially if main memory is stinted.

In the next section, we give a brief overview of related work in the area of both off-line and on-line index construction, summarizing existing techniques and discussing the shortcomings of our previous hybrid index maintenance method. Section 3 contains a description of the new method. This is followed by an experimental evaluation of old and new index maintenance strategies in section 4. Finally, we give a theoretical analysis of the new hybrid method, including a proof that, under a Zipfian term distribution, the hybridization of Logarithmic Merge provides an amortized per-posting index maintenance disk complexity of $O(1)$.

2. RELATED WORK

In this section, we give a summary of existing results in the area of off-line and on-line index construction.

Inverted Files and Off-Line Index Construction

Inverted files have been proved to be the most efficient data structure in high-performance retrieval systems for large text collections [18]. An inverted file realizes a mapping from terms to their posting lists. A term's *posting list* (also called *inverted list*) is a list of all occurrences of that term. An inverted file is a collection of posting lists, stored on a storage medium supporting random access. It is equipped with some search data structure (usually a search tree) that can be used to find the posting list associated with a given term. An example is shown in Figure 1.

Off-line index construction methods for static text collections usually follow a collection partitioning approach that splits the whole collection into a set of smaller subcollections, where the size of the individual subcollections is determined by the amount of main memory in the system. After an inverted file has been created for each such subcollection, all these sub-indices are combined in a multiway merge process, yielding the final index representing the entire collection. [9] [17] [5]

In the context of this paper, all inverted lists contain full positional information, i.e. the exact locations of all occurrences of a given term (as opposed to mere document numbers). Inverted lists are split up into chunks containing $\approx$ 30,000 postings, and each chunk is compressed using a byte-aligned gap compression method [10], resulting in an average storage requirement of $\approx$ 12 bits per posting.

The off-line process described above can be transformed into an on-line method. Search queries are then processed by fetching posting lists from all sub-indices created so far and concatenating them. Query processing performance of this strategy is very poor, of course, because all lists are heavily fragmented, implying a great number of disk seeks during query processing. We use this strategy as the baseline for index maintenance performance and refer to it as No Merge (because the inverted files are only merged after the whole collection has been indexed).

An important property of inverted files is that, as long as posting lists are stored in lexicographical order within the inverted file, the search data structure does not need to explicitly contain the position of each term's posting list. For example, if we know the positions of the list for the terms "impaired" and "impolite", then we can easily find all postings for "implicit" by reading all on-disk data between "impaired" and "impolite". If the unexplored area between two such terms is fairly small (upper limit in our implementation: 128 KB), this can be done very efficiently and does not pose a performance problem. If we allow list relocations, we lose this implicit information.

In-Place Index Maintenance

In-place index update schemes combine the existing on-disk inverted file with the in-memory data by appending all postings from the in-memory index to the corresponding list in the on-disk inverted file (creating a new list if necessary). To make this possible, they use overallocation strategies: Whenever a posting list is written to disk, more space is allocated than is actually needed. This space can then be filled with postings from the in-memory index when the system runs out of memory the next time. From time to time, of course, lists have to be relocated in order to create more room for new postings at the end of the list.

In general, there are two types of in-place update strategies: Those that require on-disk posting lists to be contiguous [8] and those that do not [14]. Forcing posting lists to be contiguous minimizes the number of disk seeks necessary to fetch a posting list (and thus maximizes query processing performance). Allowing posting lists to be non-contiguous, on the other hand, helps avoid list relocations. It therefore increases index maintenance performance, but deteriorates query performance, due to a greater number of disk seeks.

The main problem of in-place update with contiguous posting lists is that, due to the frequent relocation of posting lists, no pre-defined ordering on the terms in the inverted file can be guaranteed. Therefore, it is necessary to maintain an explicit vocabulary data structure, containing for every

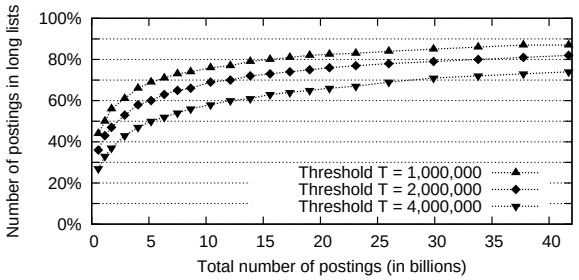

Figure 2: Total number of postings in the index vs. number of postings in long lists. After 50% of the GOV2 text collection have been indexed, more than 82% of all postings are found in lists longer than 1,000,000 postings.

term the position of its on-disk posting list. This data structure can then be used to update posting lists in the order in which they are stored on disk. Since the vocabulary can become very large (more than 50 million different terms for the text collection used in our experiments), it does not fit into main memory any more, and *vocabulary maintenance* – as opposed to *posting list maintenance* – becomes a challenging task (see [8] for details).

Merge-Based Index Maintenance

In merge-based index maintenance strategies, postings are never directly added to an existing on-disk inverted list. Instead, whenever in-memory postings have to be combined with on-disk data, the in-memory index is merged with an existing on-disk inverted file, resulting in a new inverted file that replaces the old one. The simplest merge strategy is called Immediate Merge. At any given point in time, this strategy maintains at most one active on-disk inverted file. When the indexing system runs out of memory, the in-memory data are merged with this inverted file, resulting in a new inverted file that supersedes the old one. Since every such merge operation requires the system to read the entire current on-disk index, the total amount of disk operations necessary to index a text collection containing N tokens is

$$\sum_{i=1}^{\lceil \frac{N}{M} \rceil} \Theta(i \cdot M) = \Theta\left(N \cdot \frac{N}{M}\right),$$

where M is the number of postings that can be held in memory. Despite its simplicity and its obviously problematic quadratic disk complexity, it seems to be very difficult to find an in-place update scheme that outperforms the Immediate Merge strategy (cf. Lester et al. [7] [8]).

Recently, it has been studied how allowing the indexing system to maintain more than one inverted file at a time affects the performance of merge-based update schemes. Lester et al. [6] analyzed the case where the search system is allowed to use k independent on-disk inverted files at a time (for constant k). In this situation, an optimal index maintenance strategy has an overall disk complexity of

$$\Theta\left(N \cdot \left(\frac{N}{M}\right)^{1/k}\right).$$

For $k = 2$, we refer to this strategy as Sqrt Merge.

Lester et al. [6] and Büttcher and Clarke [1] also looked at the case where the maximum allowable number of on-disk inverted files is not constant, but logarithmic in the current size of the on-disk index. The strategy proposed by Büttcher and Clarke makes use of the concept of *index generation*. An on-disk inverted file is of

- generation 0 if it has been directly created from the in-memory index;
- generation $n+1$ if it is the result of a merge operation involving indices of generation n.

Whenever there is more than one on-disk index of the same generation n, all indices of that generation are merged, resulting in a new inverted file of generation $n+1$. This process is repeated until there are no more such collisions. This strategy leads to a set of on-disk inverted files of exponentially increasing size. Its total indexing disk complexity is

$$\Theta\left(N \cdot \log \frac{N}{M}\right).$$

We refer to this strategy as Logarithmic Merge.

Hybrid Index Maintenance

Hybrid index maintenance is motivated by the fact that, for large text collections, the vast majority of all postings is found in very long posting lists containing several million entries (as shown in Figure 2). Copying these very long lists during re-merge operations (as required by Immediate Merge, for example), is very costly and should be avoided. Even though more modern strategies, like Sqrt Merge and Logarithmic Merge, substantially reduce the number of disk operations, the problem in principle remains the same. Therefore, it seems promising to distinguish between long and short posting lists and to update only short lists using a merge strategy, while long lists are updated in-place.

Earlier this year, we have presented a family of hybrid index maintenance strategies based on this distinction between short and long lists [2]. The basic idea is rather simple: As soon as the posting list for a given term exceeds a certain length (we refer to this as the long list threshold, denoted as T), is is declared *long* and moved from the merge-updated part of the on-disk index to the in-place-maintained part. Every posting list in the in-place part is stored in an individual file. Whenever a frequent term with *long* posting list is encountered during a merge operation, its postings are appended to the corresponding file instead of being transferred to the target index of the current merge operation. From the indexing system's point of view, each long list is stored contiguously inside its file. The actual details (contiguousness, relocations) are left to the file system implementation (in our experiments, the `ext3` file system that is part of the Linux kernel).

This technique can be combined with all three merge strategies described above, resulting in three different hybrid index maintenance strategies. Since these hybrid strategies require all *long* on-disk posting lists to be stored contiguously, we refer to them as HIM_C (Hybrid Immediate Merge), HSM_C (Hybrid Sqrt Merge), and HLM_C (Hybrid Logarithmic Merge) – where the subscript "C" indicates contiguous posting lists.

Shortcomings Hybrid Index Maintenance

The hybrid index maintenance strategies described above have several serious flaws. For the sake of brevity, we only discuss two major problems:

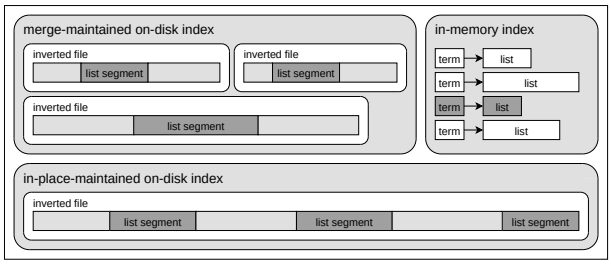

Figure 3: Index layout for a hybrid maintenance strategy with non-contiguous posting lists. Each term's posting list is the concatenation of a number of list segments found in the in-memory index, the merge-maintained index, and the in-place index. The sub-list found in the in-place part of the index may consist of several non-contiguous segments.

1. Delegating the in-place update to the file system is very convenient, but hides too much information and therefore makes it difficult to analyze the hybrid strategies. In particular, the rules that define under what circumstances a posting list is relocated (to avoid fragmentation) are unclear and may differ from system to system – even from hard disk to hard disk. Are the *long* posting lists really contiguous? Most likely not, but we don't know.

2. Assume the long list threshold of HIM_C is chosen as $T = 10^6$. After, say, 100 physical index updates, the posting list of a term X exceeds this threshold and is moved from the merge-maintained part of the on-disk index to the in-place part. From that point on, it is considered *long* and is always updated in-place. Since it took this list 100 re-merge operations to reach the threshold, on average we can expect to see 10^4 occurrences of the term between two physical index updates. For only 10^4 new postings, however, it would certainly be more efficient to store them in the merge-updated part of the index, avoiding the additional disk seek(s) caused by appending them to the end of the file associated with the term X. With every additional merge operation, this effect gets stronger, implying that the performance of the strategies described above is highly sensitive to the amount of main memory available.

In this paper, we remedy these problems. We present a new family of hybrid strategies that is less sensitive to memory limitations and more amenable to a formal complexity analysis, allowing us to get a deeper understanding of the subtleties of index maintenance for dynamic text collections.

3. A NEW SET OF HYBRID STRATEGIES

Instead of enforcing a strict distinction between short and long posting lists, as done in our previous approach to hybrid index maintenance, the new strategies allow each posting list to consist of two parts – a long one, updated in-place (realized by a single, append-only inverted file), and a short one, maintained using one of the merge strategies. Only when the length of the short, merge-maintained part of the posting list exceeds a pre-defined threshold value T, it is removed from the merge part of the index and added to the in-place part of the index. This method is similar to the

```
@addfile /u3/gov2/uncompressed/GX169/40.txt
@rank[bm25][docid][count=20][id=479] "<doc>.."</doc>" by \
    "porche", "suv"
@addfile /u3/gov2/uncompressed/GX135/81.txt
@rank[bm25][docid][count=20][id=3219] "<doc>.."</doc>" by \
    "volkswagen", "beetle", "convertible"
@addfile /u3/gov2/uncompressed/GX082/90.txt
@rank[bm25][docid][count=20][id=10406] "<doc>.."</doc>" by \
    "problems", "hmong", "immigrants"
@addfile /u3/gov2/uncompressed/GX073/87.txt
@rank[bm25][docid][count=20][id=48490] "<doc>.."</doc>" by \
    "lee", "county", "clerk", "courts"
```

Figure 4: Commands #53,437 - #53,445, taken from the whole sequence consisting of 27,204 update commands and 27,204 queries.

pulsing technique described by Cutting and Pedersen [4]. The resulting index layout is depicted in Figure 3.

The new hybrid strategies start like any of the non-hybrid merge strategies described in section 2 (Immediate Merge, Sqrt Merge, or Log. Merge). During a merge operation, however, whenever a term is encountered for which there are more than T postings participating in the merge operation, all these postings are moved to the in-place part of the index instead of the target inverted file of the merge operation. This is done by simply appending the postings to the single inverted file that represents the in-place part.

In contrast to the contiguous hybrid strategies discussed in the previous section, our new strategies introduce additional non-contiguities in the on-disk posting lists (because new postings are simply appended to the existing inverted file), causing additional disk seeks at query time. By choosing a large enough value for T, however, these query-time disk seeks can be amortized over a longer, sequential read operation If, for example, we choose $T = 10^6$, and the hard drive can read 50,000 postings (sequentially) in the time it requires to perform a single random disk seek, then the relative slowdown caused by the additional disk seeks is at most $\frac{50{,}000}{1{,}000{,}000} = 5\%$. By choosing different threshold values T, it is possible to control index maintenance and query processing performance. Smaller T values mean better update performance (because more postings are moved to the in-place part of the index). Greater T values mean better query performance (by reducing the number of disk seeks during query processing). In particular, $T = \infty$ represents a pure merge-based strategy (e.g., Immediate Merge), while $T = 0$ is equivalent to the No Merge strategy – both create the same amount of fragmentation in the on-disk posting lists. The exact effect of a given T value depends on hard drive characteristics. By measuring both the disk's bandwidth and its seek latency, it is possible to find the right T value for the relative slowdown tolerated. [1]

Since, in the in-place part of the on-disk index, posting list segments are not stored in any particular order, we need an additional data structure that tells us for every term in the inverted file the location of all its list segments. This is similar to the problem we described when we discussed the shortcomings of in-place update in section 2. This time,

[1]It needs to be mentioned here that the criterion that defines when postings are transferred to the in-place index is slightly grubby. In our system, all postings are compressed and thus have different sizes, between 1 and 6 bytes, depending on the term's frequency and locality. This should be taken into account. However, we decided to ignore this detail because it would make the analysis of our method more complicated.

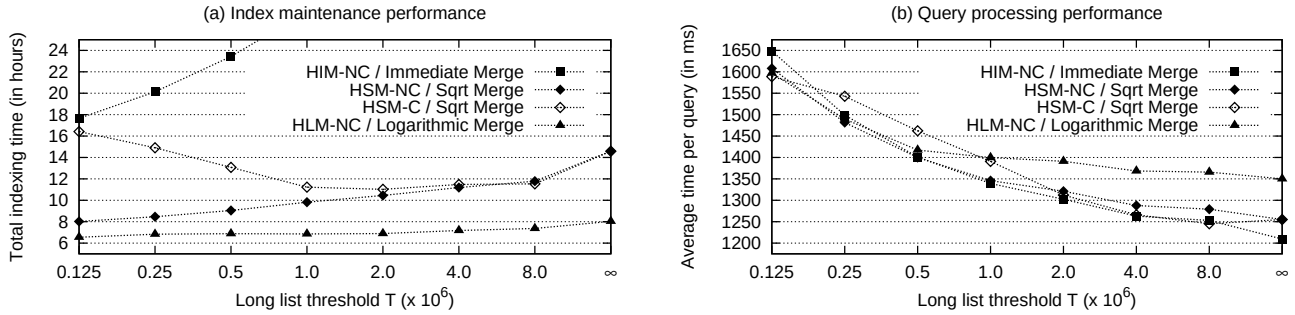

Figure 5: Index maintenance and query processing performance for different strategies with various parameter settings. $T = \infty$ represents the underlying non-hybrid strategy. $T = 0$ is equivalent to the No Merge strategy.

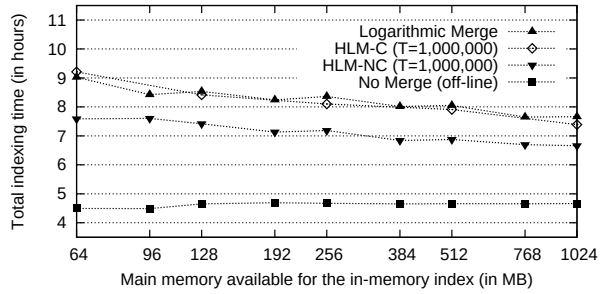

Figure 6: Impact of memory size on index maintenance performance. HLM_C is more sensitive to the amount of available main memory than HLM_NC and non-hybrid Logarithmic Merge.

however, the situation is different. Only frequent terms can make it into in-place index, and their number is very small (only 13,309 terms appear more than 10^5 times in the GOV2 text collection, for instance). We can therefore afford to keep the meta-information that is necessary to efficiently find all list segments for a frequent term in memory.

Depending on which merge strategy serves as starting point for the respective hybrid strategy, we refer to the new strategy as HIM_NC (Immediate Merge), HSM_NC (Sqrt Merge), or HLM_NC (Logarithmic Merge). In each case, the subscript "NC" indicates non-contiguous posting lists

4. EXPERIMENTAL EVALUATION

Our experiments were conducted using the GOV2 text collection used in the TREC Terabyte track[2]. GOV2 consists of 25.2 million documents with a total size of 426 GB. In order to be able to measure index maintenance and query processing performance at the same time, we created a mixed update/search sequence consisting of 27,204 update operations and 27,204 search queries (randomly taken from the 50,000 queries used in the effiency task of the 2005 TREC Terabyte track), simulating an on-line search environment. A short subsequence is shown in Figure 4. Search queries are of the form "find all documents containing at least one of the query terms, rank them by their BM25 score, and return the document IDs of the top 20 documents". All experiments were run on a single PC based on an AMD Athlon64 3500+ CPU with 2 GB of main memory and 7,200-rpm SATA hard drives. The input documents were read from a RAID-0 array built on top of two 7,200-rpm hard drives. The size of the final index, containing full positional information for all terms appearing in GOV2, was 61 GB.

[2]http://www-nlpir.nist.gov/projects/terabyte/

Indexing and Query Processing Performance

In our first series of experiments, we had our retrieval system process the command sequence, building an index for GOV2 and concurrently processing 27,204 search queries. We allowed the system to use 512 MB of main memory for the in-memory index. For the 3 different merge strategies and various threshold values T, we analyzed index maintenance and query processing performance. From the results shown in Figure 5, it is obvious that our new strategies exhibit significantly better indexing performance than the non-hybrid methods. For $T = 10^6$, HLM_NC builds the index 17% faster than its non-hybrid counterpart (HSM_NC: 49%; HIM_NC: 155%). On the other hand, query performance only drops by 4% (HSM_NC: 7%; HIM_NC: 10%). Compared to No Merge, $\text{HLM}_\text{NC}(T = 10^6)$ with 512 MB RAM only needs 47% longer to build the final index (6.86 hours instead of 4.66), but exhibits a vastly superior query processing performance – 1.4 seconds per query, instead of 4.8 seconds.

For the non-contiguous hybrid strategies, decreasing T consistently improves indexing performance and decreases query performance in all cases. Moreover, there is a strict ordering between the individual strategies. For instance, in order for HSM_NC to outperform Logarithmic Merge's indexing performance on GOV2, it needs $T \leq 2 \cdot 10^5$. At that point, however, its query processing performance is much worse than that of Logarithmic Merge (cf. Figure 5).

Space vs. Time Trade-offs

Our second series of experiments serves the purpose of finding out how the amount of RAM available for the in-memory index affects the index maintenance performance of our strategies. We therefore varied the amount of memory available for the in-memory index between 64 MB and 1,024 MB. The results depicted in Figure 6 show that, while the indexing performance of HLM_NC is changed only insignificantly, HLM_C suffers severely from the reduced size of the in-memory index, and index construction time steps up from 7.39 hours (1024 MB) to 9.21 hours (64 MB) – a 25% increase. For comparison, indexing time with HLM_NC only increases by 14% (Logarithmic Merge: 18%).

5. COMPLEXITY ANALYSIS

In this section, we generalize the experimental results presented in this paper in order to make them applicable to

other text collections. We present a complexity analysis, based on the number of disk operations (measured by the number of postings transferred to/from disk) performed by the respective strategy. More specifically, we determine the number of disk write operations necessary to index a text collection of a given size. Since the number of read operations carried out during merge operations is upper-bounded by the number of write operations (nothing is read twice), this allows us to calculate the total number of disk operations performed (up to a constant factor).

We show that the HLM_{NC} method only needs $\Theta(N)$ disk operations to construct an index for a text collection of size N. Asymptotically, this is the optimal indexing performance and is only rivalled by the No Merge strategy used in off-line index construction (which provides incompetitive query performance if used in an on-line environment). Our analysis is based on the assumption that the term distribution can be expressed as a generalized Zipf distribution [16], i.e., that the number of times the i-th most frequent word appears in the text collection is inversely proportional to i^{α}:

$$f_T(i) = \frac{c}{i^{\alpha}}$$

(rounded to the nearest integer), for some constants c and α (Zipf originally proposed $\alpha = 1$). The same assumption has been made before for similar purposes (see, for example, Cutting and Pedersen [4]). Figure 7 suggests that, for the GOV2 collection, we have a Zipf exponent α somewhere in the neighborhood of 1.2. While our analysis does not depend on the fact that the term distribution is exactly Zipfian (using a Zipf-Mandelbrot distribution [13] would lead to the same result), it does rely on the convergence of the sum:

$$\sum_{i=1}^{\infty} \frac{1}{i^{\alpha}}.$$

We therefore assume $\alpha > 1$. The sum's value is given by Riemann's Zeta function $\zeta(\alpha)$ and can be approximated by

$$\gamma + \int_1^{\infty} \frac{1}{x^{\alpha}}\, dx = \gamma + \frac{1}{\alpha - 1}, \tag{1}$$

where γ is the Euler-Mascheroni constant [15] ($\gamma = 0.5772\ldots$). For realistic Zipf exponents ($\alpha < 1.4$), the relative approximation error is less than 1%. We will therefore always use the integral representation when we refer to the value of the Zeta function. Also, we will denote the term $\gamma + \frac{1}{\alpha-1}$ from (1) as γ_{α}^{*}. In order to achieve

$$N = \sum_{i=1}^{\infty} \frac{c}{i^{\alpha}} \approx c \cdot \left(\gamma + \int_1^{\infty} \frac{1}{x^{\alpha}}\, dx\right) = c \cdot \left(\gamma + \frac{1}{\alpha - 1}\right)$$

and make the term distribution function consistent with the size of the text collection N, we let

$$c := \frac{N}{\gamma_{\alpha}^{*}} \quad \text{and thus} \quad f_T(i) := \frac{N}{\gamma_{\alpha}^{*} \cdot i^{\alpha}}. \tag{2}$$

We now use this term distribution to determine for a text collection consisting of N tokens the number of postings that are found in long posting lists (i.e., in lists that contain more than T postings, as defined in section 3). This result is then applied twice, once to analyze the disk complexity of Hybrid Immediate Merge and then a second time to analyze the disk complexity of Hybrid Logarithmic Merge.

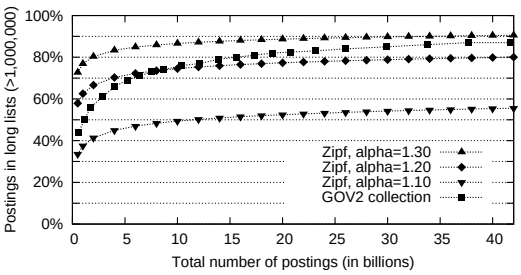

Figure 7: Comparing the term distribution of the GOV2 collection with Zipf distributions for various values of α. Zipf with $\alpha = 1.2$ seems to provide a lower bound for the total number of postings found in lists that are longer than 1,000,000 elements.

We first determine for a collection of size N the number of terms whose posting lists are longer than T postings, denoted as $L(N,T)$. From the definition of $f_T(i)$, we know:

$$f_T(i) \geq T \quad \Leftrightarrow \quad i^{\alpha} \leq \frac{N}{\gamma_{\alpha}^{*} \cdot T} \quad \Leftrightarrow \quad i \leq \left(\frac{N}{\gamma_{\alpha}^{*} \cdot T}\right)^{1/\alpha}.$$

Hence, we have

$$L(N,T) = \left\lfloor \left(\frac{N}{\gamma_{\alpha}^{*} \cdot T}\right)^{1/\alpha} \right\rfloor \in \Theta\left(\frac{N^{1/\alpha}}{T^{1/\alpha}}\right). \tag{3}$$

The total number of postings found in these lists is

$$\begin{aligned}
\sum_{i=1}^{\lfloor L(N,T) \rfloor} f_T(i) &\approx \frac{N}{\gamma_{\alpha}^{*}} \cdot \left(\gamma + \int_1^{L(N,T)} x^{-\alpha}\, dx\right) \\
&= \frac{N}{\gamma_{\alpha}^{*}} \cdot \left(\gamma + \left(\frac{(L(N,T))^{1-\alpha}}{1-\alpha} - \frac{1}{1-\alpha}\right)\right) \\
&= N - \frac{N}{\gamma_{\alpha}^{*} \cdot (\alpha - 1)} \cdot \left(\frac{N}{\gamma_{\alpha}^{*} \cdot T}\right)^{\frac{1-\alpha}{\alpha}} \\
&= N - N^{1/\alpha} \cdot \frac{T^{(1-1/\alpha)}}{(\alpha - 1)(\gamma_{\alpha}^{*})^{1/\alpha}}.
\end{aligned} \tag{4}$$

It immediately follows that the total number of postings found in short lists, i.e. in lists shorter than T postings, is

$$\hat{S}(N,T) := \frac{N^{1/\alpha} \cdot T^{(1-1/\alpha)}}{(\alpha - 1)(\gamma_{\alpha}^{*})^{1/\alpha}} \in \Theta\left(T^{(1-1/\alpha)} \cdot N^{1/\alpha}\right). \tag{5}$$

In other words, the relative number of postings found in short lists converges to zero, as N – rhythmically and inexorably – is marching towards infinity. The speed of convergence depends on the Zipf parameter α.

In the remainder of this section, we refer to the total number of postings found in short lists as $\hat{S}(N,T)$, to the number of postings found in long lists as $\hat{L}(N,T)$, and to the number of long lists as $L(N,T)$.

An Analysis of Hybrid Immediate Merge

Consider the Hybrid Immediate Merge strategy with non-contiguous posting lists (HIM_{NC}), as defined in section 3. In order to index a text collection containing N postings, HIM_{NC} needs to perform $\lceil \frac{N}{M} \rceil$ merge operations, where M is the number of postings that can be stored in main memory. In every such merge operation, HIM_{NC} moves all long posting lists (containing at least T postings) it encounters to the in-place-maintained part of the on-disk index. For

the merge-updated part of the on-disk index, which is the result of this merge operation, this means that it does not contain any lists that are longer than T postings.

Now, consider the merge-updated part of the on-disk index that results from the k-th merge operation, i.e., that is created after $k \cdot M$ tokens have been added to the collection. This inverted file contains two types of lists:

- Genuine short lists, each having $\leq T$ postings.
- Short parts of long lists, each having $\leq T$ postings.

The latter happens if, for instance, the posting list for a term was moved from the merge-maintained to the in-place-maintained part of the index in the $(k-1)$-th re-merge operation and the number of postings accumulated for that term between the $(k-1)$-th and the k-th merge operation is smaller than T so that the hybrid strategy leaves these new postings in the merge-updated part of the index.

We can easily give a lower bound for the number of postings $\hat{P}$ that are stored in the merge part of the index after the k-th merge operation:

$$\hat{P}(k \cdot M, T) \;\geq\; \hat{S}(k \cdot M, T) \;\in\; \Omega\left(T \cdot \left(\frac{k \cdot M}{T}\right)^{1/\alpha}\right)$$

(using equation 5). We can also give an upper bound:

$$\begin{aligned} \hat{P}(k \cdot M, T) \;&\leq\; \hat{S}(k \cdot M, T) + T \cdot L(k \cdot M, T) \\ \in\;& O\left(T^{(1-1/\alpha)} \cdot (k \cdot M)^{1/\alpha}\right) + O\left(T \cdot \frac{(k \cdot M)^{1/\alpha}}{T^{1/\alpha}}\right) \end{aligned}$$

(using equations 3 and 5). It follows that

$$\hat{P}(k \cdot M, T) \;\in\; \Theta\left(T^{(1-1/\alpha)} \cdot (k \cdot M)^{1/\alpha}\right).$$

Of course, this is also the number of disk write operations necessary to create the particular index. Since, for a text collection of size N, the number of disk operations spent on adding postings to the in-place part of the index is

$$\hat{L}(N, T) \in \Theta(N),$$

the total number of disk operations needed to build an inverted index for such a text collection is:

$$\begin{aligned} & \Theta(N) \qquad \text{(in-place part)} \\ +\; & \sum_{k=1}^{\lceil \frac{N}{M} \rceil} \Theta\left(T^{(1-1/\alpha)} \cdot (k \cdot M)^{1/\alpha}\right) \qquad \text{(merge part)} \\ =\; & \Theta\left(T^{(1-1/\alpha)} \cdot \frac{N^{(1+1/\alpha)}}{M}\right). \end{aligned} \tag{6}$$

For $\alpha = 1.2$ and constant T, this gives us an index maintenance disk complexity of $\Theta(\frac{N^{1.833}}{M})$, clearly better than the $\Theta(\frac{N^2}{M})$ complexity of non-hybrid Immediate Merge.

An Analysis of Hybrid Logarithmic Merge

Finally, let us consider the Hybrid Logarithmic Merge strategy with non-contiguous posting lists (HLM_{NC}), as defined in section 3. Like every index maintenance strategy discussed in this paper, HLM_{NC} accumulates postings in main memory until there is no more memory available, at which point an on-disk sub-index is created from the in-memory data. After M tokens have been added to the collection, two on-disk inverted files are created:

- a merge-maintained inverted file (with contiguous posting lists) containing $\hat{S}(M, T)$ postings;
- an in-place-maintained inverted file containing $\hat{L}(M, T)$ postings.

We refer to the merge-maintained inverted file, created after M tokens have been added, as an index of generation 0.

After the first on-disk inverted file has been created, whenever $2^k \cdot M$ tokens have been added to the collection (for some integer k), we have to merge two merge-maintained sub-indices of generation $k-1$ into a new merge-maintained index of generation k. This is the standard procedure of Logarithmic Merge. Here, however, because we follow a hybrid strategy, all long lists encountered during this merge operations are moved to the in-place part of the index instead of the new merge-maintained inverted file of generation k. Using the same argument as before (for HIM_{NC}), we can give a Θ-bound for the number of postings $\hat{P}$ in this new merge-maintained inverted file of generation k:

$$\hat{P}(2^k \cdot M, T) \;\in\; \Theta\left(T^{(1-1/\alpha)} \cdot (2^k \cdot M)^{1/\alpha}\right). \tag{7}$$

Therefore, the total number of disk write operations $D(k)$ performed during merge operations, before and including the creation of this new inverted file of generation k, is:

$$\begin{aligned} D(k) \;&=\; \hat{P}(2^k \cdot M, T) + 2 \cdot D(k-1) && (8) \\ &=\; \sum_{i=0}^{k} 2^{k-i} \cdot \hat{P}(2^i \cdot M, T) && (9) \\ &\in\; 2^k \cdot \sum_{i=0}^{k} 2^{-i} \cdot \Theta\left(T^{(1-1/\alpha)} \cdot (2^i \cdot M)^{1/\alpha}\right) && (10) \end{aligned}$$

(using equation 7). Hence, we have:

$$\begin{aligned} D(k) \;&\in\; \Theta\left(T^{(1-1/\alpha)} \cdot 2^k \cdot M^{1/\alpha} \cdot \sum_{i=0}^{k} (2^{\frac{1}{\alpha}-1})^i\right) && (11) \\ &=\; \Theta\left(T^{(1-1/\alpha)} \cdot 2^k \cdot M^{1/\alpha}\right) && (12) \\ &\subseteq\; O\left(T^{(1-1/\alpha)} \cdot 2^k \cdot M\right). && (13) \end{aligned}$$

For a text collection of size $N = 2^k \cdot M$, the number of disk operations spent on adding postings to the in-place part of the index is $\hat{L}(N, T) \in \Theta(N)$. Thus, the total number of disk operations needed to build an inverted index for a text collection of size N is:

$$\begin{aligned} & \Theta(N) \qquad \text{(for the in-place part)} \\ +\; & O\left(T^{(1-1/\alpha)} \cdot N\right) \qquad \text{(for the merge part)}. \end{aligned}$$

Since T is a constant, this means we need $\Theta(N)$ disk operations. HLM_{NC}'s disk complexity is asymptotically optimal.

Validation

In order to convince the suspicious reader of the correctness of the results obtained in this section, we use them to predict how changing the amount of available main memory M and the threshold value T affects the overall index maintenance performance of our system. We assume that the GOV2 collection, consisting of 42 billion tokens, obeys Zipf's law with $\alpha = 1.25$.

HIM_{NC}, with $T = 2 \cdot 10^6$ and 512 MB of RAM, needs 32.46 hours to index the GOV2 collection. Since the No

MERGE strategy, using the same amount of main memory, indexes the collection in 4.66 hours, the index maintenance overhead of $\text{HIM}_{\text{NC}}(T = 2 \cdot 10^6)$ is 27.80 hours. According to equation 6, altering T by a factor x changes the total index maintenance overhead by a factor $x^{1-1/\alpha}$. Thus, the expected total indexing time is:

- $27.80 \cdot 2^{0.2} + 4.66 = 36.59$ hours for $T = 4 \cdot 10^6$,
- $27.80 \cdot (\frac{1}{2})^{0.2} + 4.66 = 28.86$ hours for $T = 1 \cdot 10^6$, and
- $27.80 \cdot (\frac{1}{4})^{0.2} + 4.66 = 25.73$ hours for $T = 0.5 \cdot 10^6$.

The experimentally obtained index construction times are 37.42, 27.61, and 23.43 hours, respectively – close to the predicted numbers.

HLM_{NC}, with $T = 10^6$ and 512 MB of RAM, needs 6.87 hours to build the index, performing 150 merge operations ($k \approx 7$). The overhead, compared to NO MERGE, is 2.21 hours. Decreasing the available amount of memory to 256 MB decreases M by a factor of 2 and increases k by 1. According to equation 12, this increases the index maintenance overhead by a factor

$$2^1 \cdot (\frac{1}{2})^{1/\alpha} \approx 1.1487,$$

from 2.21 hours to 2.54 hours. We therefore predict a total indexing time of 4.66 + 2.54 = 7.20 hours. The time measured in our experiments is 7.19 hours, very close to our prediction. Similarly, for 1024 MB of main memory, we predict a total index construction time of 6.58 hours. Again, this is close to the experimentally obtained time, 6.66 hours.

6. CONCLUSION & FUTURE WORK

We have presented a new family of hybrid index maintenance strategies to be used in on-line index construction for growing text collections. Like previous hybrid strategies, they are based on a distinction between long and short posting lists. Our experimental evaluation has shown that the new strategies outperform previous index maintenance strategies, including existing hybrid strategies – which do not work very well if only a small amount of memory is available for the in-memory index.

We have given a theoretical analysis of the disk complexity of our hybrid strategies, proving that the combination of LOGARITHMIC MERGE (for short lists) and in-place update with non-contiguous list segments (for long lists) results in an overall index maintenance disk complexity of $O(N)$ for a text collection of size N. This is asymptotically optimal.

The main shortcoming of the new strategies is their slightly reduced query processing performance, caused by internal fragmentation in the on-disk posting lists. We think that this problem can be solved by integrating overallocation strategies (and possibly relocation strategies) into the in-place part and will further investigate in this direction.

7. REFERENCES

[1] S. Büttcher and C. L. A. Clarke. Indexing Time vs. Query Time Trade-offs in Dynamic Information Retrieval Systems. In *Proceedings of the 14th ACM Conf. on Information and Knowledge Management (CIKM 2005)*, Bremen, Germany, November 2005.

[2] S. Büttcher and C. L. A. Clarke. A Hybrid Approach to Index Maintenance in Dynamic Text Retrieval Systems. In *Proceedings of the 28th European Conference on Information Retrieval (ECIR 2006)*, London, UK, April 2006.

[3] T. Chiueh and L. Huang. Efficient Real-Time Index Updates in Text Retrieval Systems. Technical report, SUNY at Stony Brook, NY, USA, August 1998.

[4] D. R. Cutting and J. O. Pedersen. Optimization for Dynamic Inverted Index Maintenance. In *Proceedings of the 13th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 405–411, New York, USA, 1990.

[5] S. Heinz and J. Zobel. Efficient Single-Pass Index Construction for Text Databases. *Journal of the American Society for Information Science and Technology*, 54(8):713–729, 2003.

[6] N. Lester, A. Moffat, and J. Zobel. Fast On-Line Index Construction by Geometric Partitioning. In *Proceedings of the 14th ACM Conference on Information and Knowledge Management (CIKM 2005)*, Bremen, Germany, November 2005.

[7] N. Lester, J. Zobel, and H. E. Williams. In-Place versus Re-Build versus Re-Merge: Index Maintenance Strategies for Text Retrieval Systems. In *Proceedings of the 27th Conference on Australasian Computer Science*, Darlinghurst, Australia, 2004.

[8] N. Lester, J. Zobel, and H. E. Williams. Efficient Online Index Maintenance for Text Retrieval Systems. *Information Processing & Management*, 42, July 2006.

[9] A. Moffat and T. C. Bell. In-Situ generation of Compressed Inverted Files. *Journal of the American Society of Information Science*, 46(7):537–550, 1995.

[10] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of Inverted Indexes for Fast Query Evaluation. In *Proceedings of the 25th ACM SIGIR Conference on Research and Development in Information Retrieval*, 2002.

[11] W.-Y. Shieh and C.-P. Chung. A Statistics-Based Approach to Incrementally Update Inverted Files. *Inf. Processing and Management*, 41(2):275–288, 2005.

[12] K. A. Shoens, A. Tomasic, and H. García-Molina. Synthetic Workload Performance Analysis of Incremental Updates. In *Proceedings of the 17th ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

[13] Z. K. Silagadze. Citations and the Zipf-Mandelbrot's Law. *Complex Systems*, 11:487, 1997.

[14] A. Tomasic, H. García-Molina, and K. Shoens. Incremental Updates of Inverted Lists for Text Document Retrieval. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*, pages 289–300, New York, USA, 1994.

[15] E. W. Weisstein. Euler-Mascheroni Constant. From MathWorld – http://mathworld.wolfram.com/Euler-MascheroniConstant.html.

[16] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, USA, 1949.

[17] J. Zobel, S. Heinz, and H. E. Williams. In-Memory Hash Tables for Accumulating Text Vocabularies. *Information Processing Letters*, 80(6):271–277, 2001.

[18] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted Files versus Signature Files for Text Indexing. *ACM Trans. on Database Systems*, 23(4):453–490, 1998.