Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Information Processing and Management 41 (2005) 275–288

INFORMATION PROCESSING & MANAGEMENT

www.elsevier.com/locate/infoproman

A statistics-based approach to incrementally update inverted files

Wann-Yun Shieh [*], Chung-Ping Chung

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, ROC

Received 26 April 2003; accepted 28 October 2003
Available online 4 December 2003

Abstract

Many information retrieval systems use the inverted file as indexing structure. The inverted file, however, requires inefficient reorganization when new documents are to be added to an existing collection. Most studies suggest dealing with this problem by sparing free space in an inverted file for incremental updates. In this paper, we propose a run-time statistics-based approach to allocate the spare space. This approach estimates the space requirements in an inverted file using only a little most recent statistical data on space usage and document update request rate. For best indexing speed and space efficiency, the amount of the spare space to be allocated is determined by adaptively balancing the trade-offs between reorganization reduction and space utilization. Experiment results show that the proposed space-sparing approach significantly avoids reorganization in updating an inverted file, and in the meantime, unused free space can be well controlled such that the file access speed is not affected.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Information retrieval; Inverted file; Incremental update; Statistical approach; Spare space

1. Introduction

For most information retrieval systems, one important challenge is that a large document collection requires a specialized indexing structure. Equally important is that such an indexing structure requires an efficient incremental update mechanism.

[*] Corresponding author.
E-mail addresses: wyshieh@csie.nctu.edu.tw (W.-Y. Shieh), cpchung@csie.nctu.edu.tw (C.-P. Chung).

0306-4573/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ipm.2003.10.004

1.1. Current methods and problems

An indexing structure used by many IR systems is the inverted file (Zobel, Moffat, & Ramamohanarao, 1998; Witten, Moffat, & Bell, 1999). In an inverted file, for each distinct word (also known as a "term") in the text collection, there is a corresponding list (called the inverted list) of the form $\langle \text{term}; f_{\text{term}}; D_0, D_1, D_2, \ldots, D_{f_{\text{term}}-1} \rangle$, where identifier D_i indicates the document that contains the *term*, and frequency f_{term} indicates the total number of documents in which the *term* appears (Witten et al., 1999). Additionally, the location of each occurrence of the term in the document (say word position) may be stored with the identifier. When a user sends a request containing some query terms to an IR system, the system searches for these query terms in the inverted file to see which documents satisfy the request, and returns these documents' identifiers with word positions to the user. Zobel et al. (1998) showed that in terms of the querying time, used space, and functionality, inverted files perform better than other indexing structures.

The inverted file, however, does not support efficient incremental updates (Brown, Callan, & Croft, 1994). When new documents are added to an existing collection, the inverted lists of the terms appearing in those documents must be updated, ideally incrementally, by appending the new documents' identifiers with word positions to the tails of the lists. This update process is difficult for an inverted file because the inverted lists in the file are typically laid out sequentially and contiguously on disk with no free space between each other (Brown et al., 1994; Shoens, Tomasic, & Garcia-Molina, 1994). Any increase in length of an inverted list requires complex storage relocation and expensive free-space management.

Most conventional IR systems update the inverted file by periodically re-indexing the entire collection or by periodically merging the old, dated inverted file with the new, batched inverted files for newly arrived documents (Hsu & Lang, 1999). However, as the rate of new document arrivals grows rapidly in most applications today, rebuilding or merging the inverted files becomes too expansive and inefficient (Hsu & Lang, 1999; Ester, Kohlhammer, & Kriegel, 2000; King, 1992).

Sparing free space at the end of each inverted list for future expansion has been proposed (Brown et al., 1994; Tomasic, Garcia-Molina, & Shoens, 1994). In Brown et al. (1994), they deal with the word position inverted lists, and the sizes of the allocated spare space are determined by powers of 2 bytes (e.g., $2^4, \ldots, 2^{13}$), whereas in Tomasic et al. (1994), the sizes are determined by a constant number or by the multiple of current list length (e.g., 1.5×, 2×). In case the spare space of an inverted list is used up, a larger space is allocated and the contents of the old list are removed to the new space; the frequency of relocations can hence be reduced. Both of these approaches, however, result in much wasted space in an inverted file, and also poor performance in information retrieval.

In fact, the size of the spare space allocated for each inverted list cannot be determined easily due to a complex trade-off between relocation reduction and space utilization. If too much spare space is allocated, the possibly wasted space enlarges the inverted file and slows down the file accesses. Conversely, if the spare space is insufficient, frequent relocations cause high update costs. The best policy allocates the spare space for each inverted list according to individual space requirement.

1.2. Research goal

In this paper, we propose a statistics-based approach to allocate the spare space for an inverted list when it is relocated. This approach is based on the estimation of the space requirement in a time window. The time window for an inverted list is defined as the time interval between two sequential relocations; that is, from the time a spare space is allocated to the time the list needs to be relocated again. Whenever a time window exhausts, a suitable size of the spare space for the next allocation is predicted based on the statistics of the space usage and document update request rate in this time window. The objective of the prediction is to best guarantee that an inverted list has sufficient reserved space to amortize relocation frequency, and also to keep space utilization high. Experiment results show that the proposed space-sparing approach significantly avoids reorganization for an inverted file, and in the meantime, the wasted space can be well controlled such that the performance of file accesses would not be affected.

This paper is organized as follows. In Section 2, we model the relocation frequency and space utilization in this inverted file problem, and show how to allocate the spare space for a growing inverted list. In Section 3, we present the experiment results. Finally, Section 4 presents our conclusions.

2. Allocating spare space for a growing inverted list

To study the spare space allocation problem for a growing inverted list, we use two variables: relocation frequency and wasted space, to model the update cost and space utilization, respectively. By using these two variables, the trade-offs in determining the size of the spare space for an inverted list can be clarified. Note that in this section we deal only with document-level inverted lists that do not contain word position information for simplicity. Later in Section 3, we will show how to apply the proposed technique to word position inverted lists.

2.1. Relocation frequency and wasted space

For a growing inverted list, relocation frequency represents how often the relocation occurs, and wasted space represents how much allocated space is unused over time. Fig. 1 shows an example of relocation occurrences and space usage in the *i*th time window (see the definition in Section 1.2) for an inverted list. In Fig. 1(a), the horizontal axis represents time, and we assume

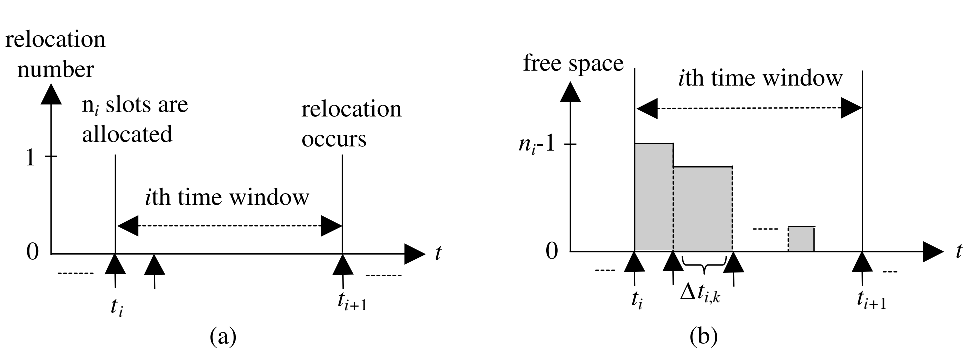


Fig. 1. For a growing inverted list: (a) the relocation occurrences in the *i*th time window, (b) the space usage in the *i*th time window.

that the ith time window starts from t_i to t_{i+1}. A vertical mark represents a relocation occurrence. If RF_i denotes the relocation frequency in the ith time window, we have

$$\mathrm{RF}_i = \frac{1}{t_{i+1} - t_i}. \tag{1}$$

In Fig. 1(b), the horizontal axis represents time, and $\Delta t_{i,k}$ represents the time between the kth and the $(k+1)$th identifier arrivals in the ith time window. The vertical axis represents unused free space, which starts from $n_i - 1$ down to 0. Without loss of generality, we assume that the unit of n_i is the number of slots and each slot stores an identifier. The identifier arriving at t_i triggers the relocation, causing n_i slots to be allocated for the list growth. At this time, the first identifier is placed in the first slot, and the spare space remained is $n_i - 1$ slots until the second identifier arrives. Each incoming identifier is placed in a slot, causing the spare space to be used up after the n_ith arrival. The next relocation occurs at t_{i+1} when there is no free space left and the next incoming identifier has arrived. Let WS_i denote the accumulated free space (i.e., wasted space) over the ith time window, we have

$$\mathrm{WS}_i = (n_i - 1) \times \Delta t_{i,1} + (n_i - 2) \times \Delta t_{i,2} + \cdots + 1 \times \Delta t_{i,n_i-1} = \sum_{k=1}^{n_i-1} (n_i - k) \times \Delta t_{i,k}. \tag{2}$$

Assume that identifiers arriving to an inverted list in a time window follow a Poisson process with rate λ. The Poisson process is often used to model a sequence of events that happen randomly and independently with a rate over time (Ross, 1996). Note that the identifier arrival rate (λ) used here is different from the document arrival rate which is the rate of documents being added into a collection. When a new document is added into the collection, the lengths of some inverted lists will increase but others will not. Thus the identifier arrival rate for a term's inverted list is proportional to the document arrival rate multiplied by the probability of the term appearing in a new document. This is what a Poisson process would model because the events of a term appearing in two documents are independent. Under the Poisson process, it is well known that the inter-arrival time is exponentially distributed.

By the Poisson distribution, the *expected relocation frequency* derived from Eq. (1) is

$$E[\mathrm{RF}_i] = \frac{\lambda_i}{n_i}, \tag{3}$$

where λ_i (arrivals/s) denotes the rate of identifiers being added to an inverted list over the ith time window. Similarly, the *expected wasted space* in the ith time window can be expressed as

$$E[\mathrm{WS}_i] = \sum_{k=1}^{n_i-1} (n_i - k) \times E[\Delta t_{i,k}] = \frac{(n_i^2 - n_i)}{2} \times \frac{1}{\lambda_i} \tag{4}$$

by taking expected values on both sides of Eq. (2). From Eqs. (3) and (4), if we can predict λ_i and assign reasonable values to $E[\mathrm{RF}_i]$ and $E[\mathrm{WS}_i]$ at the start of the ith time window, then the value of n_i can be determined at the same time.

To give reasonable values to λ_i, $E[\mathrm{RF}_i]$ and $E[\mathrm{WS}_i]$, we collect the statistics from the last two time windows for prediction. The statistics include the identifier arrival rates, relocation frequency and wasted space over the $(i-2)$th and $(i-1)$th time windows.

For λ_i, we assign a predicted value, λ'_i, to it by

$$\lambda'_i = \max(\lambda_{i-1} + \Delta\lambda_{i-1,i-2}, \lambda_c), \tag{5}$$

where $\Delta\lambda_{i-1,i-2} = \lambda_{i-1} - \lambda_{i-2}$, and λ_c is a threshold of the low bound. In (5), if the identifier arrival rate is increasing (or decreasing) between the last two time windows, i.e., $\Delta\lambda_{i-1,i-2} > 0$ (or <0), it is assumed that the arrival rate will continue to increase (or decrease) by the same amount. In case the predicted arrival rate drops to below the threshold, i.e., $\lambda_{i-1} + \Delta\lambda_{i-1,i-2} \leqslant \lambda_c$, the arrival rate is assumed to be λ_c in the next time window.

For $E[\mathrm{RF}_i]$ and $E[\mathrm{WS}_i]$, we assign predicted values, $E[\mathrm{RF}_i]_\mathrm{p}$ and $E[\mathrm{WS}_i]_\mathrm{p}$ respectively, to them by

$$E[\mathrm{RF}_i]_\mathrm{p} = \min(\mathrm{RF}_{i-1}, \mathrm{RF}_{i-2}), \tag{6}$$

and

$$E[\mathrm{WS}_i]_\mathrm{p} = \min(\mathrm{WS}_{i-1}, \mathrm{WS}_{i-2}). \tag{7}$$

Note that $E[\mathrm{RF}_i]$ is inversely proportional to n_i (see Eq. (3)). If the identifier arrival rate tends to increase in the ith time window, assigning a smaller value to $E[\mathrm{RF}_i]$ (as shown in Eq. (6)) will result in either $n_i > n_{i-1}$ or $n_i > n_{i-2}$. This makes RF_i likely to be reduced in the future. On the other hand, $E[\mathrm{WS}_i]$ is proportional to $(n_i^2 - n_i)$ (see Eq. (4)). If the identifier arrival rate tends to decrease in the ith time window, assigning a smaller value to $E[\mathrm{WS}_i]$ (as shown in Eq. (7)) will result in either $n_i < n_{i-1}$ or $n_i < n_{i-2}$. This makes WS_i likely to be reduced in the future.

There are three reasons to predict λ_i, $E[\mathrm{RF}_i]$, and $E[\mathrm{WS}_i]$ based on statistics collected in the last two time windows. First, only one time window is not sufficient for prediction because each observation point in this model is assumed to be the time that a relocation occurs, and we need at least two time windows to observe the trends of identifier arrivals. Second, collecting more recent statistics can help to measure the space requirement more accurately. From the experimental data, we found that three or earlier time windows seem not very helpful for prediction; using dated information is even harmful. We think the reason is that the popularities of some terms appearing in a collection do not always follow a regular distribution. In fact, the problem of how many previous time windows should be used for a good prediction is involved in the research scopes of the statistical method and forecasting (Thomopoulos, 1980). We did not include those detail experimental data in this paper for clarity. Finally, considering the prediction complexity (includes additional storage cost and computation), we suggest that two time windows are very suitable for prediction. In the next section, we will present how n_i can be determined by using these predicted values.

2.2. Determining n_i

By applying Eqs. (5)–(7) to Eqs. (3) and (4), we have two extreme values of n_i, say $n_{i,(3)}$ and $n_{i,(4)}$, respectively. The first value $n_{i,(3)}$ is derived from the consideration of reducing relocation frequency, and the latter $n_{i,(4)}$ is derived from the consideration of reducing wasted space. To determine n_i, we define a weighted function

$$n_i = \alpha \cdot n_{i,(3)} + \beta \cdot n_{i,(4)}, \tag{8}$$

where the values of α and β are determined by $f(\alpha,\beta)$. $f(\alpha,\beta)$ should be chosen according to the user needs; one of its simple forms may be $\alpha+\beta=1$. For systems with intensive arrivals of database updates, we suggest that $\alpha \geqslant \beta$ to favor larger n_i for reducing the frequency of updating inverted lists. Contrarily, for systems with intensive arrivals of information retrieval, we suggest that $\alpha < \beta$ to favor smaller n_i for reducing the time of retrieving inverted lists.

To implement the approach described above, an inverted list is structured as

$$\langle \mathrm{term}; f_{\mathrm{term}}; t_{\mathrm{s}}; n_{\mathrm{s}}; n_{\mathrm{r}}; \mathrm{WS_a}; \lambda_{\mathrm{p}}; \mathrm{RF_p}; \mathrm{WS_p}; D_1,\ldots,D_{f_{\mathrm{term}}}, S_{f_{\mathrm{term}}+1},\ldots,S_{f_{\mathrm{term}}+n_{\mathrm{s}}}\rangle,$$

where the additional fields are shown in Table 1. These additional fields are used to store the statistical data from the $(i-1)$th and $(i-2)$th time windows. With these data, all variables in Eqs. (3)–(8) can be derived.

Fig. 2 shows the algorithm to insert a document identifier (*id*) into an inverted list (*inv_list*). In Fig. 2, *Insert*(*inv_list*, *id*) calls three functions:

(1) ***Create***(*inv_list*, t_{now}): creates a single-slot empty inverted list.
(2) ***Space_allocate***(*inv_list*, t_{now}): calculates λ_i', $E[\mathrm{RF}_i]_{\mathrm{p}}$, $E[\mathrm{WS}_i]_{\mathrm{p}}$, $n_{i,(3)}$, and $n_{i,(4)}$, and then allocates a spare space of size n_i for *inve_list* by Eq. (8), where

Table 1
The added fields in an inverted list

New field	Description
t_{s}	Starting time of the current time window
n_{s}	Size of spare space allocated at t_{s}
n_{r}	Size of spare space remaining
$\mathrm{WS_a}$	Accumulated wasted space until now
λ_{p}	Arrival rate in the previous time window
$\mathrm{RF_p}$	Relocation frequency in the previous time window
$\mathrm{WS_p}$	Accumulated wasted space in the previous time window
S_i	Spare space

```
Algorithm Insert(inv_list, id)
begin
1       $t_{now} \leftarrow time()$;
2       if (inv_list is null)
3           Create(inv_list, $t_{now}$);
4       else if (inv_list is full)
5           Space_allocate(inv_list, $t_{now}$);
6       put id into inv_list;
7       Update(inv_list, $t_{now}$);
end
```

Fig. 2. The algorithm *Insert*(*inv_list, id*).

$$\lambda_i' = \frac{n_s}{t_{now} - t_s},$$

$$E[\mathrm{RF}_i]_p = \min\left(\frac{1}{t_{now} - t_s}, \mathrm{RF}_p\right),$$

$$E[\mathrm{WS}_i]_p = \min(\mathrm{WS}_a, \mathrm{WS}_p),$$

$$n_{i,(3)} = \frac{\lambda_i'}{E[\mathrm{RF}_i]_p},$$

and

$$n_{i,(4)} = \frac{1 + \sqrt{1 + 8\lambda_i' E[\mathrm{WS}_i]_p}}{2}.$$

Note that for WS_a in Table 1, we accumulate the wasted space by rows upon inserting identifiers as shown in Fig. 3(b), instead of by columns (shown in Fig. 3(a)). This is because from Eq. (2), we have

$$\mathrm{WS}_i = \sum_{k=1}^{n_i-1} (n_i - k) \times \Delta t_{i,k} = \sum_{j=2}^{n_i} (t_{i,j} - t_i) \times 1, \tag{9}$$

where t_i and $t_{i,j}$ denote the starting time, and the jth identifier-arrival time, in the ith time window, respectively. Accordingly, given t_s, t_{now}, and n_s, we can obtain the correct value of WS_a.

Once a spare space is allocated, the values of λ_p, RF_p, WS_p are replaced by λ_i', $\frac{1}{t_{now}-t_s}$, and WS_a, respectively; that is, the parameters of the previous time window are replaced by those of the current time window.

(3) ***Update***(*inv_list*, t_{now}): updates n_r and WS_a (by Eq. (9)) in *inv_list* after inserting *id*.

Because all of these three functions calculate only values for variables, the time complexity of the algorithm (*Insert*(*inv_list*, *id*)) is O(1). Detailed algorithms are ignored in this paper for clarity.

In Fig. 2, when an inverted list is initially to be created (*Create*(*inv_list*, t_{now})), only a single-slot space is allocated to store the first identifier. The first relocation of this list will occur when the second identifier arrives to the list. This means that the inverted lists which contain only one identifier do not require additional space to store those statistical data in Table 1. This saving is significant for our approach because these inverted lists, in fact, occupy an essential part of all inverted lists. Other inverted lists, which contain more than one identifier, however, require

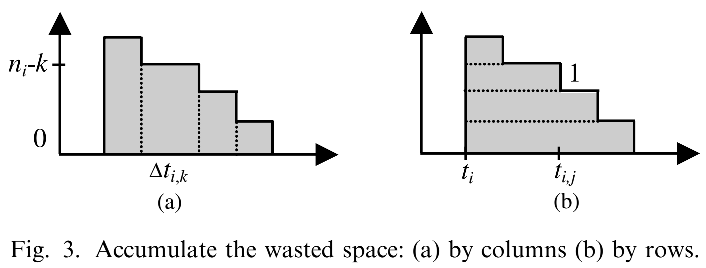


Fig. 3. Accumulate the wasted space: (a) by columns (b) by rows.

additional space to store the statistical data for spare space measurement, because in a dynamically growing document collection, some low frequency terms would possibly become fashion in the future. To avoid those additional statistical data affecting the query performance in retrieving inverted lists, we can isolate them from an inverted file, and reference them only when the relocation occurs.

3. Experiment and evaluation

The experiment is used to generate performance data. In performance evaluation, factors to be examined include relocation occurrences, storage space, and retrieval time for an inverted file.

3.1. Experiment environment

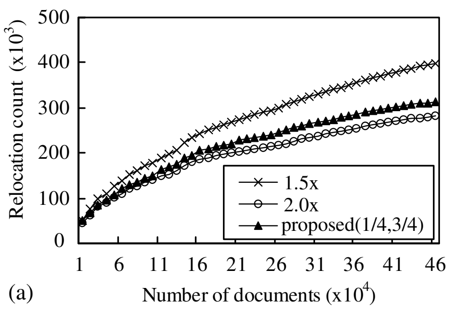
We use parts of WT10g, about 460,000 documents, to be our test collection. (WT10g is a widely distributed collection and has been included in TREC Web Test Collections (TREC, 2003).) We implement a Poisson arrival model with different document arrival rates (arrivals/s) to simulate the behavior of those documents being incrementally added into the depository. Then the proposed statistics-based approach is applied in constructing the inverted file for indexing those documents. The relocation occurrences and unused free space are monitored over time for performance evaluation.

Because the experiment purpose is to evaluate the spare space allocation in updating an inverted file, the inverted lists have not been compressed here. Witten et al. (1999) proposed many inverted file compression techniques. Most of them could be usefully incorporated into our approach, except compressing the statistical data in Table 1. Recall that these statistical data are to be used only when a relocation occurs in an inverted list. To avoid them affecting the query performance in retrieving inverted lists, we store them in an independent file, and reference this file at the time a relocation occurs. The size of this file will be taken into account as a storage space cost for our approach.

To simulate user query behavior, we implemented a query-term generator which picks query terms from the inverted files. The occurrence of these terms follows the *Zipf*-like distribution, a distribution widely used in recent IRS studies (Breslau, Cao, Fan, Phillips, & Shenkerm, 1999). In this distribution, the relative probability of a request for the ith most popular term is proportional to $1/i^{\alpha}$, where $\alpha \leqslant 1$. We let $\alpha = 0.8$ in the experiments because α appears to center around 0.8 for most traces in homogeneous environments (Breslau et al., 1999). In each experiment, we generated 100,000 user queries, and the lengths of the queries were distributed evenly from one to five terms. Furthermore, we adopted the Boolean query model, in which the AND, OR, and NOT Boolean operators are uniformly inserted into the generated queries.

3.2. Experiment results

Fig. 4 shows the relocation counts and space utilization in constructing the document-level inverted lists by three approaches. The relocation count denotes the number of relocations occurred in related inverted lists when adding new documents, whereas the space utilization denotes



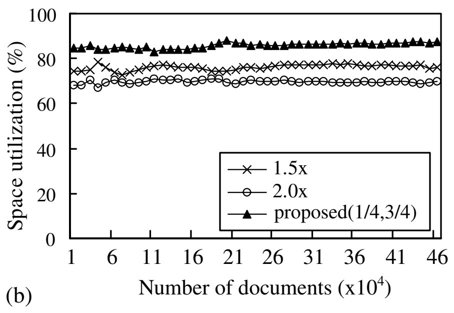


Fig. 4. Experiment results in constructing the document-level inverted file for indexing parts of WT10g: (a) relocation count, (b) space utilization.

the ratio of actual used space size to total inverted file size (containing those unused spare space and statistical data). To examine how the spare space size affects the time of retrieving an inverted list, we compare our statistics-based approach with $(\alpha, \beta) = (1/4, 3/4)$ against (1) the approach proposed in Brown et al. (1994) (denoted as "2.0×" in the figure), and (2) the approach with 1.5 times the current list length for future spare space size as proposed in Tomasic et al. (1994) (denoted as "1.5×" in the figure).

Fig. 4(a) shows that the relocation counts of the proposed approach with $(\alpha, \beta) = (1/4, 3/4)$ are somewhat higher than those of the "2.0×" approach, and are significantly smaller than those of the "1.5×" approach. On the other hand, in Fig. 4(b), the proposed approach has the highest space utilization, about 86% on average. These results show that there indeed exists a trade-off between the relocation count and the space utilization for "fixed-multiplier" approaches (e.g., 1.5×, 2×). The statistics-based approach, however, can determine the spare space size for relocation saving, or for space saving, depending on adaptive factors. When an inverted list involves fast expansion, the statistics-based approach adjusts the space allocation by relocation-saving strategy; otherwise, by space-saving strategy. This adaptability makes the proposed approach have lower relocation counts but achieve higher space utilization.

In Fig. 4(a), we essentially use the "relocation counts" to estimate the update costs. Considering that it takes rather longer to relocate a long inverted list than a short one, we use the "average number of copies per pointer in the index" to factor in this difference, as shown in Fig. 5. In the

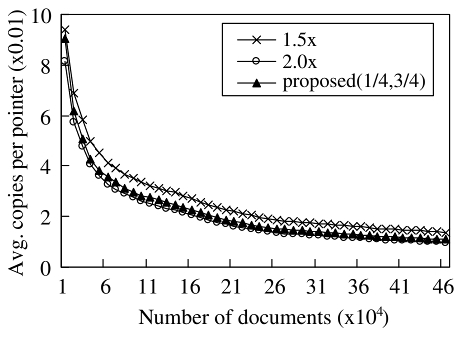


Fig. 5. Average number of copies per pointer in constructing the document-level inverted file.

figure, we find that the 1.5× approach has the largest average number of copies per pointer, and the proposed approach has it only a little larger than the 2.0× approach does. These results are consistent with the relocation counts derived from Fig. 4(a). If each copy of a pointer requires a fixed execution time, then the 1.5× approach takes the largest time to update the inverted file.

We also apply those three approaches to the word position inverted lists. Recall that in our model, relocation frequency and waste space are both related to the identifier arrival rate (λ_i) and the spare space size (n_i) in a time window (see Eqs. (3) and (4)). If we, instead, let λ_i denote the rate of "word positions" being added to an inverted list, and each word position consumes a unit of the spare space in a time window, then the technique described in Section 2 can also work well for modeling the updates in word position inverted lists. Fig. 6 shows the relocation counts and the space utilization if those three approaches are all applied to the word position inverted list updating. In Fig. 6(a), the relocation counts for all three approaches increase greatly due to the space requirements of storing word position information in inverted lists, compared with the results in Fig. 4(a). The proposed approach, however, obtains the smallest relocation counts when the number of documents increases over 260,000, and still keeps the highest space utilization (shown in Fig. 6(b)). This is because when the number of documents exceeds a scale, the fixed-multiplier approaches (e.g., 1.5×, 2×) may overestimate space requirements for some slowly growing inverted lists, but underestimate for some other fast growing inverted lists. From Figs. 4 and 6, we find that the statistics-based approach performs very well for both the document-level and word-level inverted lists.

We examine if the statistics-based approach has a stable behavior under different system loads. The system loads here are represented in terms of the document arrival rates. Given $(\alpha, \beta) = (1/4, 3/4)$ in the proposed approach, Fig. 7 shows that the document arrival rate has little effect on both performance metrics. This fact shows that the proposed approach adjusts well to different document arrival rates in estimating the size of the spare space. This can be verified by examining the derivation of n_i. Take the relocation frequency as an example. Recall that the identifier arrival rate for an inverted list is dependent on the document arrival rate. If an inverted list has an identifier arrival rate (λ_i) increasing (or decreasing) by m times, it is obvious that its RF_i will also increase (or decrease) by m times. Increasing λ_i and RF_i will immediately affect the predictions of λ'_{i+1} and $E[\mathrm{RF}_{i+1}]_\mathrm{p}$ respectively in the next time window (Eqs. (5) and (6)). This will cause n_{i+1} to be increased in response (Eq. (8)). The similar assessment also applies to the wastes space. Thus the claim stands.

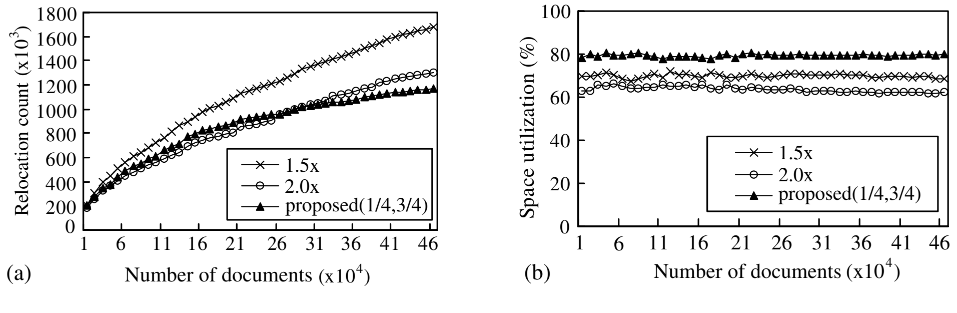


Fig. 6. Experiment results in constructing the word position inverted file for indexing parts of WT10g: (a) relocation count, (b) space utilization.

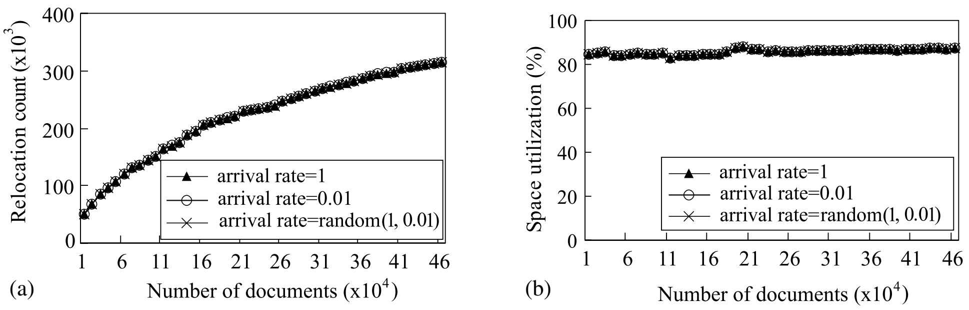


Fig. 7. Experiment results in constructing the inverted file by the statistics-based approach under different document arrival rates (arrivals/s): (a) relocation count, (b) space utilization.

Fig. 7 shows that the proposed statistics-based approach is a relocation frequency- and space utilization-stable method. What this implies is that

(1) there are no abrupt peaks of relocations happened at high database-update traffic,and
(2) no obvious space wastage at low database-update traffic.

These benefits are especially suited to the IR systems whose database-update traffic-loads are usually very dynamic and unpredictable.

We examine the experiment data, and determine if the proposed statistics-based space allocation approach has its advantages. Experiment data say that the statistics-based approach requires about 10% more relocations than the "2.0×" approach does for document-level inverted files. Although more relocations look like a disadvantage, there are more important factors to be considered. What the user really cares and sees are: the inverted file constructing time, the inverted file look-up time, and the storage space required.

Assume that the collection of 460,000 documents is ready, and we use the three approaches (1.5×, 2.0×, and statistics-based approaches) to construct three document-level inverted files for these documents. All documents are processed one by one. If an inverted list occurs a relocation, the spare space is allocated at the end of the inverted file, and its size is determined by the individual strategy of the three approaches. Without loss of generality, the freed list space (after relocation) is discarded to avoid complex reused processing. This would not affect the calculations for the storage space, because we can use the sum of all inverted list lengths to be the inverted file size. After the inverted file is generated, we use the wall-clock to measure the "inverted file constructing time" for the three approaches. Also, we use these inverted files to serve the simulated queries for measuring the "average query response time". Finally, we calculate the total size of the inverted lists, the spare space, and the statistical data to be the "storage space cost".

The proposed statistics-based approach outperforms the "1.5×" and "2.0×" approaches in all these three aspects: For the "inverted file constructing time", the ratios of the statistics-based approach versus the other two approaches are

$$\begin{aligned}&\mathit{Constructing\_Time}_{\mathit{statistics\text{-}based}} : \mathit{Constructing\_Time}_{1.5\times} : \mathit{Constructing\_Time}_{2.0\times}\\&\quad = 54.22\ \mathrm{h} : 61.67\ \mathrm{h} : 62.54\ \mathrm{h} = 1 : 1.14 : 1.15.\end{aligned}$$

While for the "average query response time", we compare the three approaches listed above with the continuous implementation of which the inverted file does not contain any spare space. The ratios of the average query response time are

$$\begin{aligned}&\mathit{Response\_Time}_{continuous} : \mathit{Response\_Time}_{statistics\text{-}based} : \mathit{Response\_Time}_{1.5\times} : \mathit{Response\_Time}_{2.0\times}\\&\quad = 5.05\text{ ms} : 5.73\text{ ms} : 6.79\text{ ms} : 7.18\text{ ms} = 1 : 1.13 : 1.34 : 1.42.\end{aligned}$$

And, finally, for the "storage space cost", Table 2 shows the relative size of the inverted file information, the spare space, and the statistical data for the three approaches. In Table 2, the inverted file information is the same for all three approaches because we use the same document collection. For the spare space, the "2.0×" approach has the highest spare space size, and the statistics-based approach has the smallest one. For the statistical data, the "statistics-based" approach requires additional space, but the "1.5×" and "2.0×" approaches do not. According to the algorithm we proposed, only 83,630 terms in the inverted file whose inverted lists contain more than one identifier require the statistical data. (The number of total terms is 214,310.) Recall that in the Table 1, there are seven variables of the statistical data required for each inverted list, and assume that each variable is stored in a two-byte integer. Then, we can derive that the total size of the statistical data for our approach equals to $83{,}630 \times 7 \times 2$, about 1.2 MB. Therefore, the ratios of the storage space cost for the three approaches compared with the continuous implementation of which the inverted file does not contain any spare space are

$$\begin{aligned}&\mathit{Space}_{continuous} : \mathit{Space}_{statistics\text{-}based} : \mathit{Space}_{1.5\times} : \mathit{Space}_{2.0\times}\\&\quad = 116\text{ MB} : 136.2\text{ MB} : 152\text{ MB} : 166\text{ MB} = 1 : 1.17 : 1.31 : 1.43.\end{aligned}$$

From the *Constructing_Time* and the *Space* ratios, we find that the proposed approach outperforms the other two approaches. For each inverted list, the trade-off between space and relocation count indeed exists—this is essentially to be the fundamental of the proposed approach. The inverted file constructing time, however, is affected not only by this trade-off but also by the total inverted file size. Recall that the inverted file constructing time is the time to construct the whole inverted file for the collection. According to our experiment, the disk seek time and disk transfer time (including the transfers of data and spare space for updating) even take a large part in the inverted file constructing time. Take a slowly growing inverted list for example. If it were allocated a larger space than actually it required, unused space during construction would affect disk accesses for other lists; though the relocation count of that list might be reduced. This is why the relocation count of the proposed approach lies between those of the other two approaches (Fig. 4(a)) but its inverted file constructing time is the smallest—the proposed approach requires the smallest storage space for the inverted file.

Table 2
Storage space cost for the three approaches

	Inverted file information	Spare space	Statistical data
1.5×	116 (MB)	36 (MB)	0
2.0×	116 (MB)	50 (MB)	0
Statistics-based	116 (MB)	19 (MB)	1.2 (MB)

All of these advantages come from the fact that while the "2.0×" or "1.5×" approach suggests a simple way of increasing the storage space for an expanding full inverted list, this simplicity may result in too generous allocations for slowly growing inverted lists, but too short-sighted allocations for other lists of the fashion terms. The experiment data show that most allocations are too generously performed. With this improperly wasted storage space, its side effect is even more devastating.

The statistics-based approach provides not only flexibility, but also stability, in spare space allocation. The flexibility is due to that each and every inverted list, upon its running out of expansion space, can be allocated new spare space tailored all for its specific needs. And the stability comes from the fact that:

1. The newly allocated spare space is determined by both the previous allocation amount, and how soon this amount was consumed. With these considerations, we are able to control the amount of allocated spare space (or space utilization in turn) and how soon we expect the next allocation to occur (or relocation frequency).
2. This space is also determined based on the size of two previous allocations. Referencing back to two time windows has the following characteristics. It gives more accurate allocation log data. It also reveals the tendency of change in allocation space requirements. While more trace-back data may be difficult to analyze and even confusing, two sets of data are very suggestive. And finally, the incurred calculation in making decisions is so simple that the overhead is negligible.

4. Conclusion

We proposed a run-time, statistics-based approach to allocate spare space in an inverted file for future updates. The approach determines the size of spare space according to the trade-offs between space efficiency and space utilization. By adaptively balancing the trade-offs, the proposed approach can incrementally update an inverted file as new documents arrive, and in the meantime, the size of unused free space can be well controlled such that the performance of file access would not be affected. The most important key point of the proposed approach is to use simple, and recently statistical data to meet the space requirements for an inverted file. This is particularly suitable for in-place updating the indexing structure of all kinds in modern large-scale IR systems, e.g., search engines, or in real-time information systems, e.g., news servers.

References

Breslau, L., Cao, P., Fan, L., Phillips, G., & Shenkerm, S. (1999). Web caching and zipf-like distributions: evidence and implications. *IEEE INFOCOM*, 126–134.

Brown, E. W., Callan, J. P., & Croft, W. B. (1994). Fast incremental indexing for full-text information retrieval. In *Proceedings of the 20th international conference of very large databases* (pp. 192–202).

Ester, M., Kohlhammer, J., & Kriegel, H.-P. (2000). The DC-tree: a fully dynamic index structure for data warehouses. In *Proceeding of the 16th international conference on data engineering* (pp. 379–388).

Hsu, W., & Lang, S.-D. (1999). Feature reduction and database maintenance in NETNEWS classification. In *Proceedings of database engineering and applications* (pp. 137–144).

King, T. (1992). *Dynamic data structure*. San Diego, CA: Academic Press, Inc.

Ross, S. (1996). *Stochastic processes*. New York, NY: John Wiley & Sons, Inc.

Shoens, K., Tomasic, A., & Garcia-Molina, H. (1994). Synthetic workload performance analysis of incremental updates. In *Proceeding of the 17th international ACM SIGIR conference on research and development in information retrieval* (pp. 329–338).

Thomopoulos, N. T. (1980). *Applied forecasting methods*. NJ: Prentice-Hall, Inc.

Tomasic, A., Garcia-Molina, H., & Shoens, K. (1994). Incremental updates of inverted lists for test document retrieval. In *Proceeding of the 1994 ACM SIGMOD international conference on management of data* (pp. 289–300).

TREC Web Test Collections (2003). Available: http://trec.nist.gov/data.html.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes—compressing and indexing documents and images* (2nd ed.). Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Zobel, J., Moffat, A., & Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems, 23*(4), 453–490.

Wann-Yun Shieh received the B.S. degree in Computer Science and Information Engineering from the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China in 1996. Currently he is pursuing the Ph.D. degree in Computer Science and Information Engineering at the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China. His research interests include computer architecture, parallel and distributed systems, and information retrieval.

Chung-Ping Chung received the B.E. degree from the National Cheng-Kung University, Tainan, Taiwan, Republic of China in 1976, and the M.E. and Ph.D. degrees from the Texas A&M University in 1981 and 1986, respectively, all in Electrical Engineering. He was a lecturer in Electrical Engineering at the Texas A&M University while working towards the Ph.D. degree. Since 1986 he has been with the Department of Computer Science and Information Engineering at the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China, where he is a professor. His research interests include computer architecture, parallel processing, and parallelizing compiler.