行政院國家科學委員會專題研究計畫　成果報告

具備動態資料庫內容與工作量管理能力之大型資訊檢索系統

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-009-083-
執行期間：91 年 08 月 01 日至 92 年 07 月 31 日
執行單位：國立交通大學資訊工程學系

計畫主持人：單智君

計畫參與人員：謝萬雲、鄭哲聖

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 92 年 10 月 23 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□期中進度報告

具備動態資料庫內容與工作量管理能力之大型資訊檢索系統設計

Dynamic Content and Workload Management for Large Scale Information Retrieval Systems

計畫類別：■ 個別型計畫　□ 整合型計畫
計畫編號：NSC 91－2213－E－009－083
執行期間：民國九十一年八月一日至民國九十二年七月三十一日

計畫主持人：單智君副教授
共同主持人：
計畫參與人員：謝萬雲(博士班)
鄭哲聖(博士班)

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學資訊工程系

中　華　民　國　九十二　年　十　月　二十一　日

中文摘要

關鍵字：資訊檢索伺服器, 資料庫更新, 工作量管理

隨著網際網路的蓬勃發展，其資訊流通的速度與使用者造成的工作量使資訊檢索系統面臨新的問題：資料庫的更新無法隨著資料本身的特性或使用者的需求而動態地調整，以及系統的資源，無法在資料更新的工作量與使用者查詢所需的工作量之間，有效地交叉利用。這使得現今的資訊檢索系統很難在不停止服務的前提下完成資料庫更新。

本計劃將以轉置檔案(inverted file)為基礎索引架構，設計一個可動態擴充的資訊檢索系統, 使其具備動態管理儲存資料 (content) 和工作量 (workload) 的機制，並在一定的效能保證下，完成資料庫更新的需求。其中的議題有:

(1) 可動態擴充的資訊儲存結構

(2) 動態配置與更新資料庫內容

(3) 儲存資料的時效性與回收策略

(4) 動態工作量分析與工作排程

英文摘要

Keywords: Information Retrieval System, database update, workload management

The rapid growth of the Internet brings new challenges for the designers of an information retrieval system. First, the system cannot dynamically update database contents according to the characteristics of data itself or to the demands of Web users. Second, the system resource cannot be fully utilized among the workload of database updates and the workload of users' retrieval. These challenges make the database update a difficult job under the condition that the system keeps serving on-line users.

We investigate an inverted-file-based information retrieval system which can be dynamically updated and expanded under the certain guarantee of retrieval-performance. In this system, we design the content manager and the workload manager with

(1) incremental data structures to store expanded data,

(2) disk placement and refreshing mechanism for the database content,

(3) identification and collection for the stale data, and

(4) dynamic job scheduling.

1. Introduction

The explosive growth in Web uses has brought new challenges for most information retrieval (IR) systems. One of the challenges is that a large document collection requires a specialized indexing structure for efficient information retrieval. Equally important is that such an indexing structure requires an efficient incremental-update mechanism.

1.1 Current methods and problems

An indexing structure used by many IR systems is the inverted file [5]. In an inverted file, for each distinct word (also known as "term") t in the text collection, there is a corresponding list (called the inverted list) of the form $< t; f_t; D_0, D_1, D_2, ..., D_{f_t-1} >$, where identifier D_i indicates the document that contains t, and frequency f_t indicates the total number of documents in which t appears. 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 to the user. Zobel et al. [5] 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 [2]. 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 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 [2]. 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. However, as the rate of new document arrival grows rapidly in most applications today, rebuilding or merging the inverted files becomes too expansive and inefficient.

Sparing free space at the end of each inverted list for future expansion has been proposed [2][3]. In [2], the sizes of the allocated free space are determined by powers of 2 bytes (e.g., 2^4, …, 2^{13}), whereas in [3], the sizes of the free space are determined by the multiple of current list length (e.g., 1.5x, 2x). In case the pre-allocated free 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 free 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 free space is allocated, the possibly wasted space enlarges the inverted file and slows down the file accesses. Conversely, if the free space is insufficient, frequent relocations cause high update costs. The best policy allocates the free space for each inverted list according to the individual space

requirement.

1.2 Research goal

In this report, we propose a statistics-based approach to allocate the free 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 free space is allocated to the time the list needs to be relocated again. Whenever a time window exhausts, a suitable size of the free space for the next allocation is predicted based on the space usage and update request rate in this time window. The goal 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. Simulation 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 report 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 simulation 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 relocation frequency and wasted space to model the update cost and space utilization.

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. Figure 1 shows an example of relocation occurrences and space usage in the *i*th time window for an inverted list. In Figure 1(a), the horizontal axis represents time, and we assume that the *i*th 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 *i*th time window for an inverted list, we have

$$RF_i = \frac{1}{t_{i+1} - t_i}. \quad (1)$$

In Figure 1(b), horizontal axis represents time, and $\Delta t_{i,k}$ represents the time between the *k*th and the (*k*+1)th identifier arrivals in the *i*th time window. The vertical axis represents unused free space, which starts from n_i down to 0. Without lost of generality, we assume that the unit of n_i is 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 free space remained is $n_i - 1$ slots until the second identifier arrives. Each incoming identifier is placed in a slot, causing the free 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

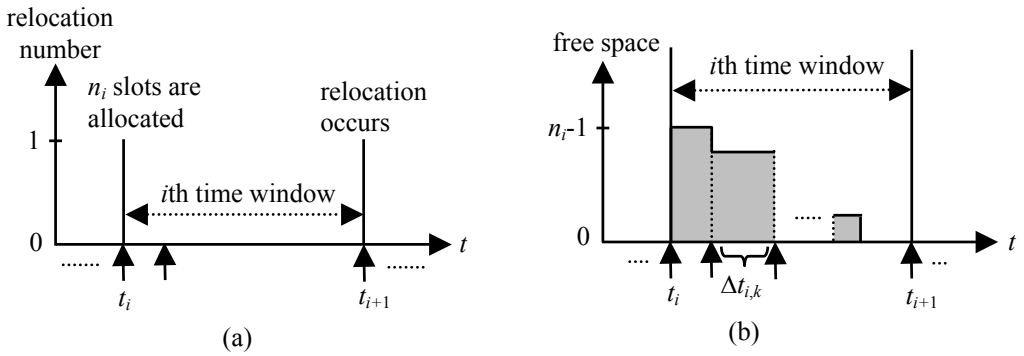


Figure 1. For a growing inverted list: (a) the relocation occurrences in the *i*th time window, (b) the space usage in the *i*th time window.

incoming identifier is arrived. Let WS_i denote the accumulated free space (i.e., wasted space) over the *i*th time window, we have

$$WS_i = \sum_{k=1}^{n_i-1} (n_i - k) \times \Delta t_{i,k} \ . \tag{2}$$

Assume that identifier arrivals in a time window follow a Poisson distribution pattern. Then the *expected relocation frequency* is

$$E[RF_i] = \frac{\lambda_i}{n_i}, \tag{3}$$

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

$$E[WS_i] = \frac{(n_i^2 - n_i)}{2} \times \frac{1}{\lambda_i} \tag{4}$$

by taking expected values on both sides of Equation (2). From Equations (3) and (4), if we can predict λ_i and assign reasonable values to $E[RF_i]$ and $E[WS_i]$ at the start of the *i*th time window, then the value of n_i can be determined at the same time.

To give reasonable values to λ_i, $E[RF_i]$ and $E[WS_i]$, we collect the statistics from 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

$$\lambda_i' = \begin{cases} \lambda_{i-1} + \Delta\lambda_{i-1,i-2} & \text{if } \lambda_{i-1} + \Delta\lambda_{i-1,i-2} > 0 \\ \lambda_{i-1} & \text{otherwise} \end{cases}, \tag{5}$$

where $\Delta\lambda_{i-1,i-2} = \lambda_{i-1} - \lambda_{i-2}$. 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. This is the case for $\lambda_{i-1} + \Delta\lambda_{i-1,i-2} > 0$. Or when the arrival rate drops to $\lambda_{i-1} + \Delta\lambda_{i-1,i-2} \le 0$, the arrival rate is assumed unchanged in the next time window.

For $E[RF_i]$ and $E[WS_i]$, we assign predicted values, $E[RF_i]_p$ and $E[WS_i]_p$ respectively, to them by

$$E[RF_i]_p = \min(RF_{i-1}, RF_{i-2}), \text{ and} \tag{6}$$

$$E[WS_i]_p = \min(WS_{i-1}, WS_{i-2}) . \quad (7)$$

Note that $E[RF_i]$ is inversely proportional to n_i (see Equation (3)). If the identifier arrival rate tends to increase in the *i*th time window, assigning a smaller value to $E[RF_i]$ (as shown in Equation (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, $E[WS_i]$ is proportional to $(n_i^2 - n_i)$ (see Equation (4)). If the identifier arrival rate tends to decrease in the *i*th time window, assigning a smaller value to $E[WS_i]$ (as shown in Equation (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 two reasons to predict λ_i, $E[RF_i]$, and $E[WS_i]$ based on statistics collected in last two time windows. First, collecting more recent data can help to measure the space requirement more accurately. In fact, the identifier arrival behavior fluctuates in the real world; using dated information is harmful for prediction. Second, using statistics from more time windows requires more storage for each inverted list. This makes the space usage inefficient. 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 Equations (5), (6), and (7) to Equations (3) and (4), we have two candidate values of n_i, say $n_{i,(3)}$ and $n_{i,(4)}$, respectively. The first candidate is derived from the consideration of reducing relocation frequency, and the latter from the consideration of reducing wasted space. To determine n_i, we define a weighted function

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

where $\alpha + \beta = 1$, and (α, β) can be determined adaptively by system demands. For the systems which have intensive arrivals of database updates, we suggest $\frac{\alpha}{\beta} \geq 1$ to favor larger n_i for reducing the frequency of updating inverted lists. Contrarily, for the systems which have intensive arrivals of information retrieval, we suggest $\frac{\alpha}{\beta} < 1$ to favor smaller n_i for reducing the time of retrieving inverted lists.

To support the approach described above, new fields are added into an inverted list as shown in Table 1. These new fields are used to store the statistical data from the (*i*-1)th and (*i*-2)th time windows. By these data and with a run-time clock t_{Now}, all variables in Equations (3)-(8) can be derived. (Due to space limit, we omit the details.)

3. Simulation and evaluation

Simulation 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.

Table 1. New fields to be added into an inverted list

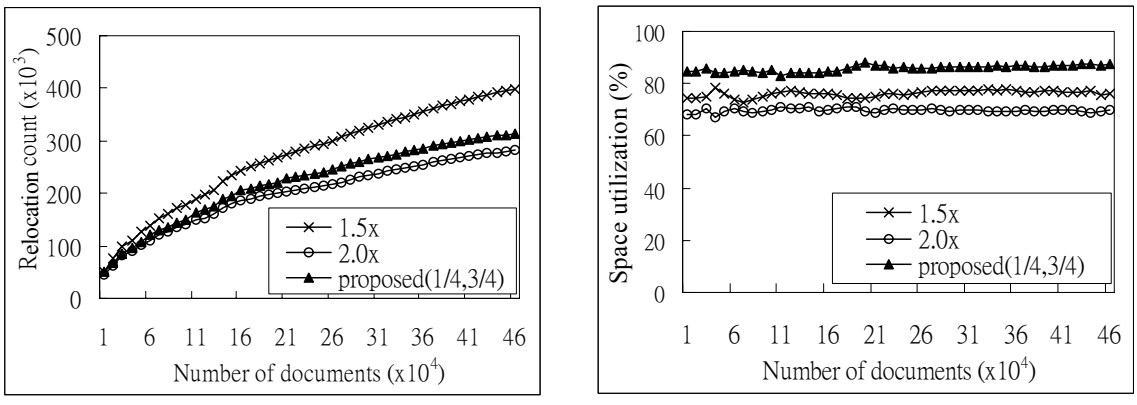
fields	Description
t_s	Starting time of the current time window
n_s	Size of spare space allocated at t_s
n_r	Size of spare space remained
WS_a	Accumulated waste space until now
λ_p	Arrival rate at the previous time window
RF_p	Relocation frequency at the previous time window
WS_p	Accumulated waste space at the previous time window
S_i	Spare space

3.1 Simulation 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 [6].) We implement a Poisson arrival model to simulate the behavior of those documents being incrementally added into the depository. Then the proposed statistics-based space sparing 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.

3.2 Simulation results

Figure 2 shows the relocation counts and space utilization in constructing the inverted files 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 the ratio of actual used space size to total inverted file size (containing those unused spare space and statistical data). To examine that the spare space affects the time of retrieving an inverted list, we compare our statistics-based approach within $(\alpha, \beta) = (1/4, 3/4)$ against the approach proposed in [2] (denoted as "2x" in the figure), and the approach within 1.5 times of current list length as



(a) (b)

Figure 2. Simulation results in constructing the inverted file for indexing parts of WT10g: (a) relocation count, (b) space utilization.

spare-space size proposed in [3] (denoted as “1.5x” in the figure).

Figure 2(a) shows that the relocation counts of our proposed approach with $(\alpha, \beta) = (3/4, 1/4)$ approximate those of the “2.0x” approach, but smaller than those of the “1.5x” approach. In Figure 2(b), however, the space utilization of the statistics-based approach is better than that of the “2.0x” approach by about 16%, and also better than that of the “1.5x” approach by about 10%. According to our simulation, the decrease in space utilization for the “2.0x” approach causes the average time of retrieving an inverted list to be increased by about 42% compared with the inverted file without spare space, and for the “1.5x” approach by about 34%. (The average retrieving time for our proposed approach increases only about 10%).

We examine the simulation data, and determine if the proposed statistics-based space allocation approach has its advantages. Simulation data say that the statistics-based approach requires about 10% more relocations than the “2.0x” approach does. Although more relocations look like a disadvantage, this is really transparent to the user. What the user really cares are: the inverted file constructing time, the inverted file look-up time, and the storage space required. The proposed statistics-based approach outperforms the “1.5x” and “2.0x” approaches in these three metrics: For the overall time of constructing an inverted file (assume the documents are processed one by one), the increasing ratios of the statistics-based approach versus the other two approaches are

$Time_{statistics\text{-}based} : Time_{1.5x} : Time_{2.0x}$

=54.22 hrs : 62.54 hrs : 61.67 hrs=1 : 1.15 : 1.14.

While for the average time of retrieving an inverted list, the speed ratios of the statistics-based approach versus the other two approaches are

$Speed_{statistics\text{-}based} : Speed_{1.5x} : Speed_{2.0x}$

$$= \frac{1}{5.61\ \text{ms}} : \frac{1}{6.79\ \text{ms}} : \frac{1}{7.18\ \text{ms}} = 1.28 : 1.06 : 1.$$

And, finally, for the storage space, the space requirement ratios of the statistics-based approach versus the other two approaches are

$Space_{statistics\text{-}based} : Space_{2.0x} : Space_{1.5x}$

=135 MB : 152MB : 166 MB=1 : 1.13 : 1.23.

All of these advantages come from the fact that while the “2.0x” or “1.5x” 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 slow growing inverted lists, but too short-sighted allocations for other lists of the fashion terms. The experiment data show that most allocations are too generous performed. With this improperly wasted storage space, its side effect is even most 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 propose 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 extractive of the proposed approach is to use simple 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 of modern large-scale IR systems, e.g., search engines, or in real-time information systems, e.g., news servers.

5. References

[1] T. King, *Dynamic Data Structure*, Academic Press, Inc. 1992.

[2] E. W. Brown, J. P. Callan, W. B. Croft, "Fast Incremental Indexing for Full-Text Information Retrieval," *Proc. of the 20th International Conference of Very Large Databases*, Sep. 1994, pp. 192-202.

[3] A. Tomasic, H. Garcia-Molina, K. Shoens, "Incremental Updates of Inverted Lists for Test Document Retrieval," In *R. T. Snodgrass and M. Winslett, editors, Proceedings of the 1994*.

[4] K. Shoens, A. Tomasic, H. Garcia-Molina, "Synthetic workload performance analysis of incremental updates," *Proc. of the 17th Inter. ACM SIGIR Conf. on Research and Development in Information Retrieval*, July 1994, pp. 329-338.

[5] J. Zobel, A. Moffat, K. Ramamohanarao, "Inverted Files Versus Signature Files for Text Indexing," *ACM Transactions on Database Systems*. Vol. 23, No. 4, 1998, pp. 453-490.

[6] TREC Web Test Collections, http://trec.nist.gov/data.html

[7] S. Ross, *Stochastic Processes*, John Wiley & Sons, Inc. 1996.

[8] Witten, I. H., Moffat, A., and Bell, T.C., *Managing Gigabytes - Compressing and Indexing Documents and Images*. 2nd Ed., Morgan Kaufmann Publishers, Inc. 1999.

[9] M. Ester, J. Kohlhammer, H.-P. Kriegel, "The DC-Tree: A Fully Dynamic Index Structure for Data Warehouses," Proceeding of the 16th International Conference on Data Engineering, 2000, pp. 379-388.