NTCIR-3 WEB Experiments at Osaka Kyoiku University

— Towards Index Partitioning and Parallel Retrieval —

Takashi SATO Yukikazu KYO Kihei KOBATA
Osaka Kyoiku University
4-698-1 Asahigaoka, Kashiwara, Osaka, Japan
sato@cc.osaka-kyoiku.ac.jp

Abstract

Long gram-based indices are experimented at NTCIR-3 WEB task. To make gram-based indices, no analyses such as morphological ones are required. 2 byte characters extracted from NTCIR-3 'cooked' version of WEB task corpus. The total index size is 26 Gbyte and time to make indices is about 18 hours. Median search time per word from index is 197msec. Ranking algorithm used is based on a traditional probabilistic model. We report index partitioning which we experimented. And we propose parallel retrieval.

Keywords: *gram-based index, gram coding, index partitioning, parallel retrieval, NTCIR*

1 Introduction

Retrieval using search engines is one of most common way to find target WEB pages. And there are many well-known search engines in the world. NTCIR-3 WEB task is notable because it regards WEB search as retrieval task. Since the size of NTCIR-3 WEB task corpus is very large, this task is interesting from the point of realization itself of large retrieval system.

When we retrieve web pages, we can find words in mind well using full text retrieval systems than keyword retrieval systems. Among full text retrieval systems, systems whose indices are based on suffix array [1-5] or grams [6-10] are effective, we think, since every character sequences which include words, compound word, etc. are retrievable. In making indices, they need no dictionary and no morphological analyses.

In order to make suffix array efficiently, we have to put corpus on computer main memory. So one problem of suffix array is that we can not make indices for big corpus.

The size of corpus for NTCIR-3 WEB is 10 and 100Gbyte, which is far bigger than that of former NTCIR tasks. Since this size exceeds main memory capacity of most computers, it is impossible to make suffix array indices practically.

On the other hand, general gram-based indices are thought to have following problems.

(1) The size of index becomes huge when gram length is more than 3 for Japanese corpus.

(2) Word retrieval requires not only search for grams, which compose the word, but also computation of intersection over searched sets. Then it becomes slow.

Our group has made gram-based indices [11-13] for large corpus. We report in this manuscript that our indices do not have the above problems.

When an index becomes large, index partitioning is a worthy to try approach, we think. Making, retrieving and managing index become easy by this approach. Moreover processing time will decrease considerably since it enables parallel processing. We report index partitioning, which we tried at this task.

2 Index making

We compute gram as [14], document by document. We made an index as an inverted file of gram. During index making, we sort gram. We cannot put the entire corpus in main memory at one time since the corpus of this task exceeds main memory size. We first made batch indices by internal sorting algorithm from subsets of corpus, which fit in main memory. Batch indices are merged into partitioned indices in batch order. At this task, five partitioned indices constitute the entire index For each partitioned index, we made

wide range map of gram values, which are put in main memory when we search grams.

At this task, most grams constitute 4 or 5 characters, and they are coded into w_g=6 byte. That is, a gram has almost same length as 3-gram if not coded. Computer used is Sun Blade 100 (2GB main memory).

Table 1 shows the size of corpus, *cooked* (prepared by organizer), *trimmed* (extracted 2 byte characters from cooked) and indices for 10 and 100Gbyte corpus. Index size overhead against trimmed is 173 and 165% respectively. Table 2 shows time to make indices. Numbers (0...4) in part# row below 100Gbyte indicates the number of partitioned indices.

Table 1. Size of corpus, cooked, trimmed and index

corpus	10Gbyte	100Gbyte					
cooked	4.86Gbyte	39.2Gbyte					
trimmed	1.69Gbte	15.5gbyte					
part#	total	0	1	2	3	4	total
index size	2.95Gbyte	4.6	5.6	5.4	5.1	4.9	25.6Gbyte

Table 2. Time to make index

corpus	10Gbyte	100Gbyte					
part#	total	0	1	2	3	4	total
time	3.1hr	3.2	3.9	3.7	3.5	3.3	17.7hr

3 Query making

We extract query words using morphological analysis from TITLE and/or DESK tags in given 56 topics (0008-0063). Compound words are segmented in words, and then all possible combinations of words are made of a compound word.

In this task, we set essential words [14]. Plural words can be essential in OR, considering we can set synonyms.

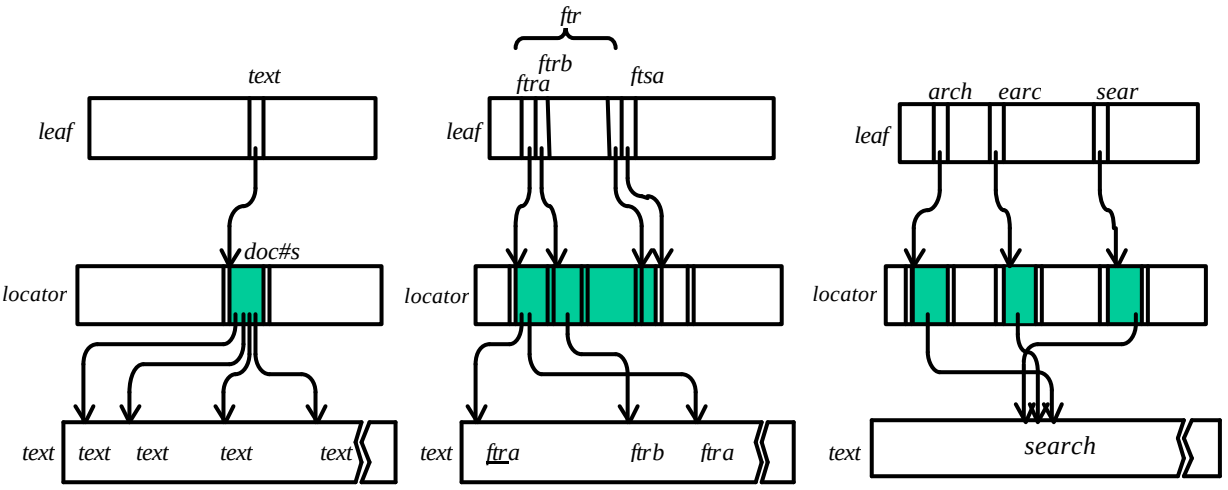
4 Index Searching and Document Ranking

Each partitioned index has three parts called *root*, *leaf* and *locator*. Grams are sorted and stored in secondary storage as leaf. There is a pointer from a gram in leaf to a bucket, which stores the document numbers where the string corresponding to the gram is found. Locator, which is stored in secondary storage, is collection of buckets. Root, which is put in main memory when searching, is a wide range map of grams.

The search algorithm is explained in terms of three cases according to the relation between the length of search key word (l_k) and gram length (l_g). Figure1 (a), (b) and (c) show how to follow the pointers in leaves and locators when $l_k = l_g$, $l_k < l_g$ and $l_k > l_g$ respectively. Since the buckets of the locator are stored sequentially, they are drawn in one box and separated by double lines.

Our index has tree structure, which has sorted gram and wide range map of them. So, not only query words whose length is equal to gram length, but also shorter or longer words can be searched efficiently. When we search a longer word, every gram in the words is searched. Then retrieved sets of document numbers are intersected

From set of retrieved documents for query words, we compute tf-idf and similarity using probabilistic model [15] for document ranking. Table 3 shows retrieval time including ranking.



(a) $l_k = l_g$ (b) $l_k < l_g$ (c) $l_k > l_g$

Figure 1. Index search

Table 3. Retrieval time

corpus	10Gbyte	100Gbyte
retrieval words	299	299x5
time to search all words	54.3sec	639sec
search time per word (average, median)	(182msec, 25.8msec)	(427msec, 197msec)
retrieval time per query	58.5sec	287sec

6 Discussions

We realized that to make large index in itself was difficult. For example, the access to a large file (>20Gbyte) is not efficient on UNIX file system. We made five sets of partitioned indices in the order of documents number. By partitioning index, not only index making but also document management becomes easier. We sought words and ranked documents set by set. Then we merged five ranked results.

Although we did not experiment at this task, the parallel processing of index making, word search and document ranking become possible. The sequential processing will remain in the merge part, however, we expect speed up of entire processing. So, parallel processing will be promising approach for large indices.

7 Conclusions

We experimented our long gram-based indices at NTCIR-3 WEB task. Since our indices are based on grams, no analyses such as morphological ones are required. We made an index from 2 byte characters extracted from NTCIR-3 'cooked' version of WEB task corpus. The total index size is 26 Gbyte and time to make indices is about 18 hours. Median search time per word from index is 197msec. Ranking algorithm used is based on a traditional probabilistic model. We report index partitioning which we experimented. And we propose parallel retrieval.

References

[1] Gonnet, G., Baeza-Yates, R. and Snider, T., New Indices for Text: Pat Trees, in *Information Retrieval: Data Structure & Algorithms* chapter 5, Frakes, W. and Baeza-Yates, R. Ed., pp. 66-82 (1992).

[2] Shang, H. and Merrett T., Trees for approximate string matching, *IEEE Trans. Knowledge and Data Eng*., Vol. 8, No. 4, pp. 540-547 (1996).

[3] Itoh, M., An Efficient Method for Constructing Suffix Arrays of Large Texts, *IPS Japan SIG Notes*, 99-NL-129-5 (1999).

[4] Yamashita, T., Fujio M. and Matsumoto Y., Language Independent Tools for Natural Language, *Proc. 18th ICCPOL*, pp.237-240 (1999).

[5] Ferragina, P. and Grossi, R., Fast string searching in secondary storage: Theoretical developments and experimental results, *Proc. ACM-SIAM Symposia on Discrete Algorithms*, Vol. 7, pp. 373-382 (1996).

[6] Ogawa, Y. and Iwasaki, M., A new character-based indexing method using frequency data for Japanese documents, *In Proc. 18th ACM SIGIR Conf.*, pp. 121-129 (1995).

[7] Sugaya, N. *et al.*, A full-text search system for large Japanese text bases using n-gram indexing method, *Proc. 53th Annual Convention IPS Japan*, 5T-2,3 (1996).

[8] Akamine, S. and Fukushima, T., Flexible string inversion method for high-speed full-text search, *Proc. Advanced Database Symposium '96* (1996).

[9] Matsui K., Namba, I. and Igata, N., Full-text searching engine for large-scale data, *Proc. 1997 IEICE General Conference*, D-4-6 (1997).

[10] Kikuchi, C., A fast full-text search method for Japanese test database, *Trans. IEICE*, Vol. J75-D-1, No. 9, pp. 836-846 (1992).

[11] Sato, T., Fast full test search with free word using TS-file, *Proc. 19th ACM SIGIR Conf.*, p.342 (1996).

[12] Sato, T., Fast full test retrieval using gram based tree structure, *Proc. ICCPOL '97*, Vol.~2, pp. 572--577 (1997).

[13] Sato, T. *et al.*, Gram based full test search system and its application, IPSJ SIG Notes, 98-DBS-114-2 (1998).

[14] Sato, T, *et al.*, NTCIR-3 PAT experiments at Osaka Kyoiku university, *in this proceedings.*

[15] Robertson, S.E. and Walker, S., Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval, *Proc. 17th Int. Conf. Research and Development in Information Retrieval*, pp. 232-241 (1994).