IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

POSITIONAL DATA ORGANIZATION AND COMPRESSION OF INVERTED INDEXES

Anuradha R. Kale

Assistant professor,
Computer science and Engineering,
P.R.Pote Patil Institute of Engineering and Research, Amravati,India.

Abstract: To sustain the tremendous workloads they suffer on a daily basis, Web search engines employ highly compressed data structures known as inverted indexes. Previous works demonstrated that organizing the inverted lists of the index in individual blocks of postings leads to significant efficiency improvements. Moreover, the recent literature has shown that the current state-of-the-art compression strategies such as PForDelta and VSEncoding perform well when used to encode the lists docIDs. . Compression reduces both the size of indexes and the time needed to evaluate queries.

In this paper we examine their performance when used to compress the positional values and revisit the compression of inverted lists of document postings that store the position and frequency of indexed terms. A) we introduce PFBC, a simple yet efficient encoding scheme, which encodes the positional data of an inverted list block by using a fixed number of bits. PFBC allows direct access to the required data by avoiding costly look-ups and unnecessary information decoding, achieving several times faster positions decompression than the state-of-the-art approaches. B) For compression of inverted lists considering two approaches to improving retrieval efficiency: better implementation and better choice of integer compression schemes.

Index Terms **- Inverted indexes, , index compression, Positional data organization.**

I. INTRODUCTION

Search engines have demanding performance requirements. Users expect fast answers to queries, many queries must be processed per second, and the quantity of data that must be searched in response to each query is staggering. The demands continue to grow: the Google search engine, for example, indexed around one billion documents a year ago and now manages more than double that figure.So, several works proposed methodologies for storing the index data in a special manner which allows us to skip large portions of the lists during query processing. These approaches suggest partitioning the inverted lists of the index in a number of adjacent blocks which can be individually accessed and decompressed.

The benefits of these methods are magnified in the case where we store positional data within the index. Inverted indexes are used to evaluate queries in all practical search engines . Compression of these indexes has three major benefits for performance. First, a compressed index requires less storage space. Second, compressed data makes better use of the available communication bandwidth; more information can be transfered per second than when the data is uncompressed. For fast decompression schemes, the total time cost of transfering compressed data and subsequently decompressing is potentially much less than the cost of transferring uncompressed data. Third, compression increases the likelihood that the part of the index required to evaluate a query is already cached in memory, thus entirely avoiding a disk access. Thus index compression can reduce costs in retrieval systems.

We introduce PFBC, a scheme which encodes the positions of an inverted list block by using a fixed number of bits allowing us to a) access the required data almost instantly and b) decode only the data actually needed, without touching any unnecessary information. We demonstrate that with a small cost in space, PFBC outperforms all the adversary compression methods in terms of speed, when applied on the positional data of the index.

In this paper, we revisit compression schemes for the inverted list component of inverted indexes. There have been a great many reports of experiments on compression of indexes with bitwise compression schemes , which use an integral number of bits to represent each integer, usually with no restriction on the alignment of the integers to byte or machine-word boundaries.

II. The Positions Fixed-Bit Compression (PFBC)

In this Section we describe PFBC, a simple, yet efficient approach for encoding and organizing the positional data of an inverted list. ster and we also decompress less data. These methods are magnified in the case where we store positional data within the index. This is due to the fact that the size of the positions is several times larger than that of docIDs and frequencies and the indexes containing positional values are about 3 to 5 times larger than the non-positional ones. Therefore, it is extremely important to devise an effective mechanism to organize and compress the positional data, since a naive solution could lead to prohibitively large indexes and reduced query throughput. In this work we demonstrate that although the current block compression methods are both effective and efficient when applied at docIDs and frequencies, they do not perform equally well when they operate upon the positional data of II an inverted list.

The fixed bit compression methodology of PFBC is expected to introduce some compression loss in comparison to PForDelta. Actually, the latter encodes the largest integers of a list as exceptions and the rest of them by using a fixedbit scheme, similar to the one we described. This operation is proved to be very effective in the case of docIDs, because in the docIDs blocks the number of large integers is small. However, when P4D is applied at blocks of positional data, the benefits are diminished because in such blocks the number of large integers cannot be predicted. Indeed, as we demonstrate by our experiments, PFBC is outperformed by P4D in terms of compressed sizes by only a small margin.

PFBC exhibits a wide range of advantages over the adversary approaches: –

- It facilitates direct access to the positional data by using equation .
- No expensive look-ups for positions in tree-like structures are required. Consequently, query processing is accelerated; –
- It saves the space cost of maintaining a separate look-up structure , since the involved pointers can be stored within the skip table;
- It uses fewer pointers than the indexed lists of Transier and Sanders ;
- It enables decoding of the information actually needed, without the need to decompress entire blocks or sub-blocks of integers.

III. INVERTED INDEXES

An inverted index consists of two major components: the vocabulary of terms—for example the words—from the collection, and inverted lists, which are vectors that contain information about the occurrence of the terms.

IV. COMPRESSING INVERTED INDEXES

Special-purpose integer compression schemes offer both fast decoding and compact storage of inverted lists. In this section, we consider how inverted lists are compressed and stored on disk. We limit our discussions here to the special-purpose integer compression techniques that have previously been shown to be suitable for index compression, and focus on their use in increasing the speed of retrieval systems. Without compression, the time cost of retrieving inverted lists is the sum of the time taken to seek for and then retrieve the inverted lists from disk into memory, and the time taken to transfer the lists from memory into the CPU cache before they are processed. The speed of access to compressed inverted lists is determined by two factors: first, the computational requirements for decoding the compressed data and, second, the time required to seek for and retrieve the compressed data from disk and to transfer it to the CPU cache before it is decoded. For a compression scheme to allow faster access to inverted lists, the total retrieval time and CPU processing costs should be less than the retrieval time of the uncompressed representation. However, a third factor makes compression attractive even if CPU processing costs exceed the saving in disk transfer time: compressing inverted lists increases the number of lists that can be cached in memory between queries, so that in the context of a stream of queries use of compression reduces the number of disk accesses. It is therefore important that a compression scheme be efficient in both decompression CPU costs and space requirements. There are two general classes of compression scheme that are appropriate for storing inverted lists. Variable-bit or bitwise schemes store integers in an integral number of bits. Well-known bitwise schemes include Elias gamma and delta coding and Golomb-Rice coding . Bytewise schemes store an integer in an integral number of blocks, where a block is eight bits in size; we distinguish between blocks and bytes here, since there is no implied restriction that a block must align to a physical byte-boundary. A simple bytewise scheme is variable-byte coding ; uncompressed integers are also stored in an integral number of blocks, but we do not define them as bytewise schemes since, on most architectures, an integer has a fixed-size representation of four bytes.

V. CONCLUSIONS

In this paper we introduced PFBC, a method especially designed for organizing and compressing the positional data in Web inverted indexes. PFBC operates by employing a fixed number of bits to encode the positions of each inverted list block, and stores a limited number of pointers which enable the direct retrieval of the positional data of a particular posting. Compared to the current state-ofthe-art compression techniques, PFBC offers improved efficiency allowing direct access without look-ups, and very fast decompression. The experiments we have performed on a 50 million document collection demonstrated that in contrast to OptP4D and VSEncoding, the proposed approach touches much fewer data and allows about 5 times faster positions decompression.

Compression of inverted lists can significantly improve the performance of retrieval systems and efficiently implemented variable-byte bytewise scheme results in query evaluation that is twice as fast as more compact bitwise schemes. Moreover, we have demonstrated that the cost of transferring data from memory to the CPU cache can also be reduced by compression: when an index fits in main memory, the transfer of compressed data from memory to the cache and subsequent decoding is less than that of transferring uncompressed data. Using byte-aligned coding, we have shown that queries can be run more than twice as fast as with bitwise codes, at a small loss of compression efficiency. These are dramatic gains.

REFERENCES

[1] Anh, V., Moffat, A.: Structured index organizations for high-throughput text querying. In: String Processing and Information Retrieval, pp. 304–315 (2006)

[2] Anh, V., Moffat, A.: Index compression using 64-bit words. Software: Practice and Experience 40(2), 131–147 (2010)

[3] Boldi, P., Vigna, S.: Compressed perfect embedded skip lists for quick invertedindex lookups. In: String Processing and Information Retrieval, pp. 25–28 (2005).

[4] Chierichetti, F., Kumar, R., Raghavan, P.: Compressed web indexes. In: Proceedings of the 18th international conference on World wide web, pp. 451–460 (2009).

[5] G. Navarro, E. Silva de Moura, M. Neubert, N. Ziviani, and R. Baeza-Yates. Adding compression to block addressing inverted indexes. Information Retrieval, 3(1):49–77, 2000