Increase the Speed of Search Index in the Duplicate Text Detection Systems

E. V. Sharapova
Murom Institute
Vladimir State University
Murom, Russia
mivlgu@mail.ru

Abstract—**The work is devoted to the organization of the search index of the duplicate text detection systems Author.NET. The paper considers the structure of the search index. The search index can be divided into several groups of files – terms, documents, TF*IDF index, signatures, shingles. In the article the ways of organizing the search index are considered – index storage in RAM, compression of index file, tree index, index table of contents. It is given a proposal for organizing a search index for duplicate text detection system. During the experiments, it was found that the most profitable option is to use compressed index files stored on the SSD.**

Keywords—search index; index; index file; duplicate detection; duplicate text detection system

I. Introduction

The search for duplicate texts is widely used at present [1]. Anti-plagiarism systems are used to verify scientific articles and student works. A systems of texts originality checking (Advego Plagiatus, Text.ru, eTxt.ru) actively used in journalism and copywriting. Email services identify similar emails as spam. News agencies group determine similar news.

The duplicate text detection systems are actively developing. In this regard, their effective and quick work becomes an important task. One of the most important elements is the effective organization of the search index. The work is devoted to the organization of the search index of the duplicate text detection systems Author.NET [2].

II. Search Index Structure

Studies have shown that using various kinds of DBMSs (MySQL, Oracle, MSSQL Server, etc.) as a search base for large collections of documents does not provide the required level of performance - the data access time is quite long. For this reason, it is proposed to use structured binary files to store search indexes [3, 4, 5, 6].

Let's consider the structure of the search index in more detail. The search index can be divided into several groups of files:

1) Terms.

The group includes a glossary of terms (words found in documents), a list of stop words (the most common words from a dictionary that are not informative and are not considered when searching) [7, 8], IDF index (inverse frequency of documents) [9, 10, 11, 12].

2) Documents.

The group includes a list of all documents entered into the system, indicating their unique identifiers, location in the system and the Internet, the language of the document, the number of words in it, belonging to a particular collection, etc., as well as the full texts of documents in Unicode text format [13].

3) TF * IDF index.

The group includes indexes of frequency values of the terms TF and TF*IDF calculated for each word in each document [14, 15].

4) Signatures.

The group includes the signatures [16, 17] calculated for each document - the CRC32 and MD5 checksum, the signatures of the six most significant words according to the values TF, TF*IDF, TF*RIDF, the signatures of the longest and most significant sentences, as well as the "long" signature of description words.

5) Shingles.

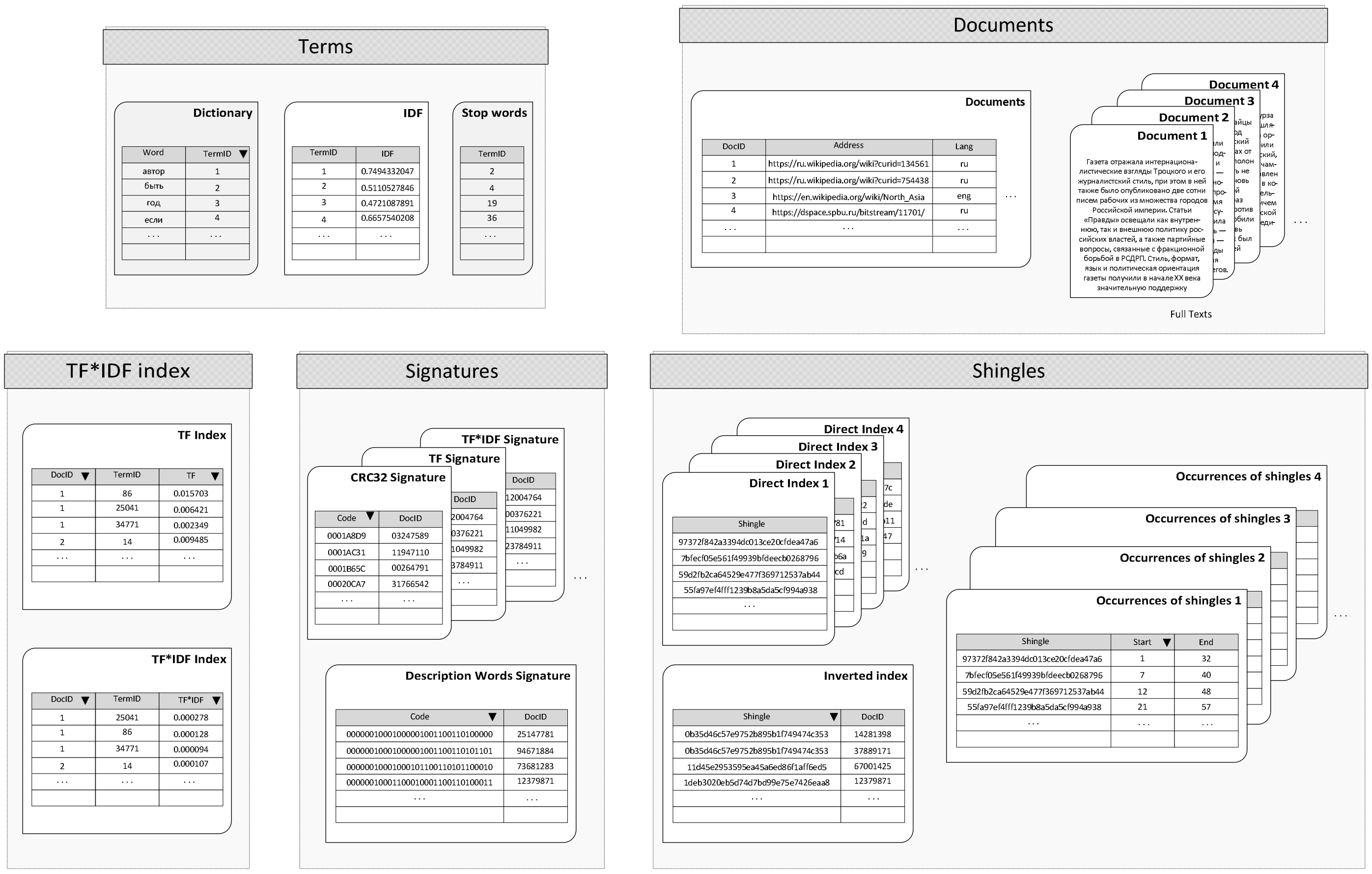
The group includes sets of direct shingle indices (arranged in the order of shingles in the document) for each document, indices of occurrence of shingles in each document (start and end positions in the text), as well as an inverse index sorted by the values of shingles [18].

III. Presentation Options for Search Index

The main problem of the indicated structure of the search index is its large size. For example, only the TF index for 2300000 documents occupies 22 GB in plain text format or 15 GB in the form of a structured digital file.

The operation of reading such an index file takes a lot of time, which affects the search time for text duplicates. The internal structure of the index files is quite simple and convenient to use. For this reason, it is advisable not to change its format. This raises the problem of trying to speed up work with index files without significantly modifying them.

978-1-7281-6072-6/20/$31.00 ©2020 IEEE



Word	TermID ▼
автор	1
быть	2
год	3
если	4
. . .	. . .

TermID	IDF
1	0.7494332047
2	0.5110527846
3	0.4721087891
4	0.6657540208
. . .	. . .

TermID
2
4
19
36
. . .

DocID	Address	Lang
1	https://ru.wikipedia.org/wiki?curid=134561	ru
2	https://ru.wikipedia.org/wiki?curid=754438	ru
3	https://en.wikipedia.org/wiki/North_Asia	eng
4	https://dspace.spbu.ru/bitstream/11701/	ru
. . .	. . .	. . .

Газета отражала интернациона-листические взгляды Троцкого и его журналистский стиль, при этом в ней также было опубликовано две сотни писем рабочих из множества городов Российской империи. Статьи «Правды» освещали как внутрен-нюю, так и внешнюю политику рос-сийских властей, а также партийные вопросы, связанные с фракционной борьбой в РСДРП. Стиль, формат, язык и политическая ориентация газеты получили в начале XX века значительную поддержку

DocID ▼	TermID	TF ▼
1	86	0.015703
1	25041	0.006421
1	34771	0.002349
2	14	0.009485
. . .	. . .	. . .

DocID ▼	TermID	TF*IDF ▼
1	25041	0.000278
1	86	0.000128
1	34771	0.000094
2	14	0.000107
. . .	. . .	. . .

Code ▼	DocID
0001A8D9	03247589
0001AC31	11947110
0001B65C	00264791
00020CA7	31766542
. . .	. . .

Code ▼	DocID
0000001000100000010011001101000000	25147781
0000001000100000010011001101011101	94671884
0000001000100010110011010110001 0	73681283
0000001000110001000110011010001 1	12379871
. . .	. . .

Shingle
97372f842a3394dc013ce20cfdea47a6
7bfecf05e561f49939bfdeecb0268796
59d2fb2ca64529e477f369712537ab44
55fa97ef4fff1239b8a5da5cf994a938
. . .

Shingle ▼	DocID
0b35d46c57e9752b895b1f749474c353	14281398
0b35d46c57e9752b895b1f749474c353	37889171
11d45e2953595ea45a6ed86f1aff6ed5	67001425
1deb3020eb5d74d7bd99e75e7426eaa8	12379871
. . .	. . .

Shingle	Start ▼	End
97372f842a3394dc013ce20cfdea47a6	1	32
7bfecf05e561f49939bfdeecb0268796	7	40
59d2fb2ca64529e477f369712537ab44	12	48
55fa97ef4fff1239b8a5da5cf994a938	21	57
. . .	. . .	. . .

Fig. 1. Search index structure.

A. *Index storage in RAM*

We can try to reduce the number of file read operations if it takes a long time to read the index file. The simplest solution is to read the index file into RAM before starting the duplicate text detection system. In the future, work does not occur with the index file itself, but with its copy in RAM [19].

The gain in access speed is obvious - the read speed of RAM is thousands of times higher than that of a hard drive. Read time from disk can be reduced when using SSD drives. But still, the speed of RAM is hundreds of times higher than SSD.

However, to store the search index in memory, it is necessary that there is enough free RAM. For 15 GB files, this becomes a significant problem.

The maximum amount of RAM with which the 64 bit version of Windows 10 Pro can work is 512 GB. Technically, we can build a system with a large amount of RAM. If the search index fits in memory, then this is the best solution.

Nevertheless, what if the search index takes 1 Terabyte? Then there is not even a physical opportunity to place the entire index in RAM.

B. *Compression of index file*

It is possible to use file compression to reduce disk space [20, 21, 22, 23, 24, 25, 26]. A smaller file will be read from disk faster. That is, the access time to the file is reduced. Instead, it takes time to unpack the information stored in the file. Thus, a gain in speed will be observed when the time to unpack the data is less than the saved time when reading data.

The time for unpacking the data substantially depends on the compression algorithm used and its software implementation. We used ZIP compression. The best speed results were obtained using ZipForge.

Using compression reduces the size of the index file by almost 5 times. Thus, the above described TF index after compression became occupy 3 GB instead of 15 GB.

Compressed index files have a significant drawback. When changing the index, it is necessary to compress it again, i.e. make changes “on the fly” will not succeed.

C. *Tree index*

Storing indexes of terms occurrence in documents or shingles is a special task. Can make a separate file for each term, including a list of all documents where it is used. Files will be significantly different in size. For frequently used terms (words), the files will have a significant size, for rare terms - only a few bytes. However, the main problem is the number of such files. With a dictionary size of 140 thousand words, the number of files will also be 140,000. When storing them in one directory, there are significant delays in processing files associated with the processing by the operating system of the table of contents of the directory (file list).

To speed up processing, it is possible to present the index in the form of a directory/subdirectory/file tree, in the form 00/00/01.idx. Then each directory contains up to 100 subdirectories or index files.

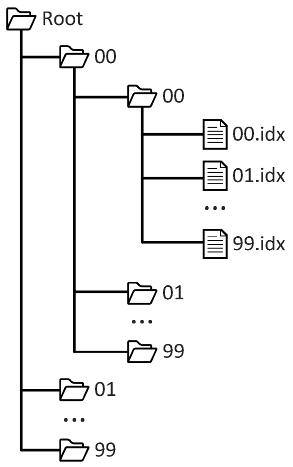


Fig. 2. Tree index structure.

However, there is another problem - the increase in real disk space required to store files. So everything, even the smallest files will occupy at least one cluster on the disk (for example, 4096 bytes).

D. *Index Table of Contents*

A table of contents can be used to speed up data access. The essence of the table of contents is as follows. Data is stored in one (several) large files [27, 28, 29]. Inside the index file, the data is grouped by some criterion (for example, the term ID).

The table of contents contains information indicating the file name, the initial position of the data in the file and the length of the information block [30]. Then only need to open the specified file, go to the specified position and read the data block of the desired length.

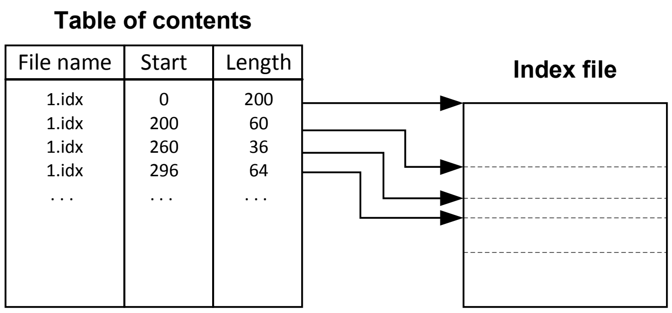


Fig. 3. Index Table of Contents.

When organizing the index of occurrence of terms in this way, can get rid of the problem of "eating" memory with small files and reducing performance due to the processing of the table of contents directories with a large number of files.

The downside is the complexity of compressing the index file. If compress the entire file, then the table of contents cannot be used without completely unpacking the file. Can compress each data block individually. However, the effect of compression will not be good enough, since the data inside the block will be compressed without taking into account the repeating sequences in other blocks.

IV. Organization of the Search Index

In the course of our experiments, the following option for working with the search index was proposed. Dictionaries of terms (words) and stop words are read at the beginning of the duplicate detection system and stored in RAM. The total data size is less than 3 MB. Also in memory is stored the index of inverse frequency of documents (IDF). Signature files are small (3 MB each) and are read into RAM at system startup.

The TF and TF-IDF indices are stored as compressed files. The size of each uncompressed part is 1 GB. The total number of files is 15.

Term entry indexes are stored as a tree index. The choice in favor of a tree-like index was made because of the ability to make changes to the index in the process of work without the need to rebuild the entire index. The number of index files is equal to the number of terms in the dictionary.

The number of direct indices of shingles and indices of occurrence of shingles is equal to the number of documents in the system - more than 2300000. For storage, it was decided to use large index files with a table of contents.

Since indexes are built on documents, when adding new documents, information is simply added to the end of the index file and the table of contents is updated. In contrast, the term entry indices change significantly when adding new documents, which makes the use of large index files with a table of contents not rational.

V. Test Results

To investigate the speed of access to the search index in their various organizations we have TF index, and spent his sequential reads. The index was placed in the form of a MySQL database (MyISAM type), in the form of structured binary files, located on the HDD and SSD. To evaluate the compression efficiency, we used compressed index files, also located on the HDD and SSD.

A 2TB SEAGATE Barracuda ST2000DM008 with SATA III interface, 256 MB buffer memory and 7200 rpm spindle speed were used as an HDD drive. A 240 GB ADATA Ultimate SU650 solid-state drive with TLC 3D NAND memory was used as an SSD drive. A 16 GB of DDR4 RAM was installed on a test computer.

During testing, 125000000 records were sequentially read from the index. The test results are shown in table I.

TABLE I. TEST RESULTS

Index storage	Time
Index in MySQL	24:23.538
Index in HDD	11.141
Index in SSD	6.792
Compressed index in HDD	8.636
Compressed index in SSD	6.694

Reading time from the database turned out to be very long - more than 24 minutes. Thus, reading the entire index from the database will take several hours. Of course, by optimizing the database, reading time can be reduced. But still, this is unacceptable a lot.

Reading time from the HDD is 11.141 seconds, and with an SSD - almost 2 times faster. When reading large sizes of index files, the SSD was 3.5 times faster than the HDD. As can see, there is a gain in speed, but it is not so significant. Using search index compression allow to speed up its processing time using HDD by 30%. When using SSD, there is practically no gain in processing speed. However, at the same time, the amount of disk space occupied by the index is significantly reduced. This is especially true for the more expensive SSD drives.

VI. CONCLUSION

The proposed version of the organization of the search index for the duplicate text detection system allow to effectively organize its work and achieve high speed to data. Storing frequently used data in RAM, compressing the search index, using the table of contents of large index files, allows to reduce the time of access to data and to achieve high speed of searching for duplicate texts. During the experiments, it was found that the most profitable option is to use compressed index files stored on the SSD.

Acknowledgment

The reported study was funded by RFBR according to the research project № 19-07-00692.

References

[1] R. Sharapov and E. Sharapova, “The problem of fuzzy duplicate detection of large texts,” CEUR Workshop Proceedings, vol. 2212, pp. 270-277, 2018

[2] E. Sharapova and R. Sharapov, “System of fuzzy duplicates detection,” Applied Mechanics and Materials, vol. 490-491, pp. 1503-1507, 2014.

[3] I.H. Witten, A. Moffat and T.C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images, Morgan Kaufmann Publishers, 1999. 550 p.

[4] S. Heinz and J. Zobel, Efficient single-pass index construction for text databases, 2003.

[5] G. Salton and M.J. McGill, Introduction to modern information retrieval. McGraw-Hill, 1983. 448 p.

[6] C.D. Manning, P. Raghavan and H. Schütze, Introduction to Information Retrieval. Cambridge University PressCambridge, England, 2009. 544 p.

[7] C. Fox, “A stop list for general text,” SIGIR Forum, vol. 24, pp. 19–21, 1990.

[8] R.T.W. Lo, B. He, and I. Ounis, “Automatically building a stopword list for an information retrieval system,” Proc. of the 5th Dutch-Belgian Information Retrieval Workshop (DIR’05), Utrecht, Netherlands, 2005.

[9] R. Blanco and A. Barreiro, “Static pruning of terms in inverted files,” Lecture Notes in Computer Science, vol. 4425, pp. 64–75, 2007.

[10] J. Zobel and A. Moffat, “Inverted files for text search engines,” ACM Computing Surveys, vol. 38, pp. 1–56, 2006.

[11] C. Buckley and A. F. Lewit, “Optimization of inverted vector searches,” SIGIR ’85: Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 97–110, 1985.

[12] D. Cutting and J. Pedersen, “Optimizations for dynamic inverted index maintenance,” Proceedings of the 13th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 405–411, 1990.

[13] S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina, “Building a distributed full-text index for the web,” ACM Transactions on Information Systems, vol. 9, pp. 217–241, 2001.

[14] G. Salton, A. Wong and C.S. Yang, “A vector space model for automatic indexing,” Communications of the ACM, vol.18, n.11, pp.613-620, 1975.

[15] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Information Processing & Management, vol. 24, n. 5, pp. 513–523, 1988.

[16] B. Carterette and F. Can, “Comparing inverted files and signature files for searching a large lexicon,” Information Processing and Management, vol. 41, pp. 613–633, 2005.

[17] R.A. Baeza-Yates and B.A. Ribeiro-Neto, Modern Information Retrieval. ACM / Addison-Wesley, 1999.

[18] Z.J. Czech, G. Havas and B.S. Majewski, “An optimal algorithm for generating minimal perfect hash functions,” Information Processing Letters, vol. 43, pp. 257–264, 1992.

[19] J. Zobel, S. Heinz, and H.E. Williams, “In-memory hash tables for accumulating text vocabularies,” Information Processing Letters, vol. 80, pp. 271–277, 2001.

[20] P. Ferragina and R. Venturini, “Compressed permuterm index,” SIGIR ’07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 535–542, 2007.

[21] J. Goldstein, R. Ramakrishnan, and U. Shaft, “Compressing relations and indexes,” ICDE, pp. 370–379, 1998.

[22] D. Blandford and G. Blelloch, “Index compression through document reordering,” DCC’02: Proceedings of the Data Compression Conference, pp. 342–351, 2002.

[23] A. Moffat and V.N. Anh, “Inverted index compression using word-aligned binary codes,” Information Retrieval, vol. 8, pp. 151–166, 2005.

[24] P. Boldi and S. Vigna, “Compressed perfect embedded skip lists for quick inverted-index lookups,” SPIRE 2005 String Processing and Information Retrieval, Lecture Notes in Computer Science, pp. 25–28, 2005.

[25] A. Moffat and L. Stuiver, “Binary interpolative coding for effective index compression,” Information Retrieval, vol. 3, pp. 25–47, 2000.

[26] W.-Y. Shieh, T.-F. Chen, J. J.-J. Shann, and C.-P. Chung, “Inverted file compression through document identifier reassignment,” Information Processing and Management, vol. 39, pp. 117–131, 2003.

[27] G. Candela and D. Harman, “Retrieving records from a gigabyte of text on a mini-computer using statistical ranking,” Journal of the American Society for Information Science, vol. 41, pp. 581–589, 1990.

[28] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer, “Static index pruning for information retrieval systems,” Proceedings of the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 43–50, 2001.

[29] E.A. Fox and W.C. Lee, “Fast-inv: A fast algorithm for building large inverted files,” Technical report, Virginia Polytechnic Institute & State University, Blacksburg, VA, USA, 1991.

[30] A. Moffat and J. Zobel, “Self-indexing inverted files for fast text retrieval,” ACM Transactions on Information Systems, vol. 14, pp. 349–379, 1996.