HECATE: A FULL-TEXT RETRIEVAL SYSTEM FOR SHORT TEXT

Song Wang[1, a], Yongping Xiong[1, b]

[1]Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China;

[a]wangsong@bupt.edu.cn, [b]ypxiong@bupt.edu.cn

Keywords: short text, full-text retrieval.

Abstract. Nowadays, the proportion of Chinese short text gradually increases in the Internet information. Without the division on the length of text, indexing and retrieval of traditional full-text retrieval system result in low efficiency. Oriented to short text application scenarios , we designs overall three steps for short text, including the information exchange, indexing, retrieving and optimizes the index storage structure, the selection of core words and eventually completes a system from input of the document to searching results output. Compared with the mainstream open source full text search engine Elasticsearch, Experimental results show that it has a great advantage in indexing time, space performance, retrieval time and it is necessary for full-text retrieval system to design and optimize in terms of short text.

Introduction

As a platform for the integration of the vast amounts of information, technology of information retrieval brings the whole world to millions of people and makes a lot of information at hand [1]. Commercial search engine that takes information retrieval as core technology has become the Internet portal. It is the focus of major Internet companies' competition, and its development has been gradually maturing. The number of search engines has reached several thousands. Yahoo!, Google, Baidu, Sogou are internationally known for a high degree among them.

Full-text retrieval is an important part of information retrieval which is not only a single technology, but also exists as a standard function of Internet product. With mature of the search engine applications , more and more open source retrieval system was developed, such as Lucene, Sphinx, Solr, Elasticsearch, Whoosh, Zebra ,covering C, C++, Java, Python and other programming languages. They have their own advantages in indexing and retrieval time and space efficiency, and have an important application in academic research and prototype construction.

When the full text retrieval system has been basically met the needs of users, the mobile Internet era arrives. The fragmentation of information attributes are gradually becoming apparent. Because of its wide range of application scenarios and the diversity of content expression, short text [2] is ubiquitous in the Internet and mobile terminals, such as micro-blog, forums, short message, reply and comment of articles, chatting records. It is playing an important role nowadays and in the foreseeable future. The characteristics of the short text on Internet including large scale, fast spread, low quality, diversities [3], as well as the characteristics of incomplete, sparse and fragmentation, will affect the full text retrieval system. In the background of vast base of short text, there is no retrieval system customized for short text. Only in a part of the full-text retrieval system, there are some solutions, such as the reference document model for short text, improvement of the estimation accuracy [4] of short text model, a fast short text retrieval strategy [5] or setting up a model based on the micro-blog search model [6]. But none of them has a whole process optimization for all short text from indexing to retrieval. There are a lot of extra room for research and optimization during short text indexing and retrieval.

In this article, we design and implement a real-time short text full-text retrieval system named Hecate with high performance, high throughput. Using principle of classification, decoupling, Parallelization. It contains totally three independent services including data receiving, indexing and retrieval. It

provides offline data input, online query receiving, retrieval and do basic sorting of results. According to the characteristics of short text, we optimize the system design in offset storage, logic of Chinese word selection and document storage and so on. Testing results show that the indexing time and space efficiency are improved in different degrees compared with the mainstream open source full-text retrieval system Elasticsearch.

System Architecture

The overall architecture is divided into three parts as shown in Figure 1, data receiving service, index service and retrieval service. Each service is relatively independent and they communicate with each other through message queue and shared memory. The basic process of the system consists of offline short-text document acceptance, establishment of forward index and inverted index, processing delete and modification of document, online real-time query, retrieval inverted index, merge and sort the results of the query.

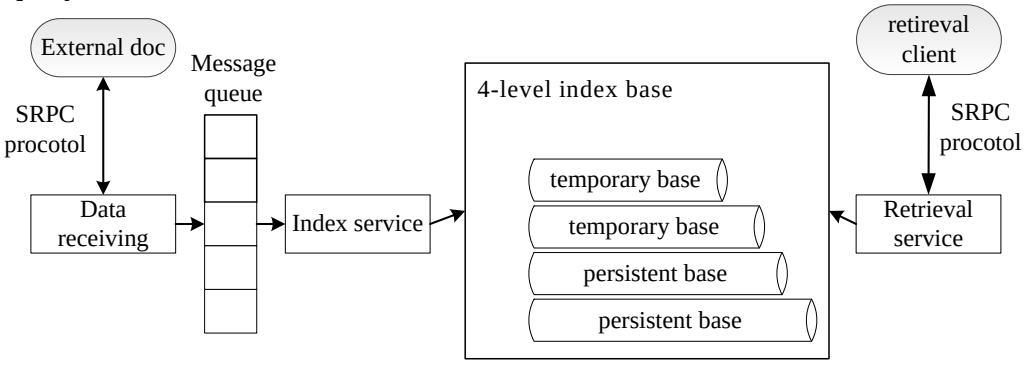

Fig. 1 overall system architecture

The data receiving service uses a simple RPC protocol to receive the documents from the outside, and then the document is parsed and forwarded to message queue. Message queue is a thread safe queue based on POSIX shared memory, which plays the role of data transfer and buffering. Because this article is aimed at the short text, a large number of scattered short text will impact on the system. Considering the characteristics of short text, we seperate data receiving service from inde service. Although the data receiving service can also be placed in index services, when the index service is decoupled, the index service is not only easy to expand and maintenance, but also preventing the blocking of its service for other components.

The indexing service is the core of the system whose purpose is to generate forward index and inverted index used for retrieval service. It also includes word segmentation and merging functions. Index service monitors message queue and reads document and establish index in parallel using multi threads. Index base uses hierarchical storage to support efficient and massive retrieval. Hecate has totally 4 index bases. They are created in merging level by level. It is merged to the next level of the index base when capacity of each base reaches a certain threshold .

Retrieval service provides online query function. For a query request, it makes query analysis, extraction of the core word, generation of query syntax tree. After the cache misses, it retrieves all of 4 - level index bases of inverted index in parallel and implements logical operation of *and ,or, not* , as well as filtering, scoring and sorting of results.

Data Receiving Service

In order to improve the efficiency of the index, the data receiving service uses I/O multiplexing and multi thread to improve the concurrency. Main thread monitors TCP port, binds network event and controls entry rate. When the document encapsulated in the SRPC protocol is sent over, main thread assigns it to several sub threads of executive function that get the data and do some simple processing, write message queue and continually receive requests.

SRPC Protocol. The protocol is a kind of RPC protocol based on TCP/IP socket which is responsible for both-way communication of retrieval and indexing between client and server. This protocol has The characteristics of a high efficiency, simpleness, extension, etc. including *head* and *body* . *head* has totally 20 bytes.

Table 1. SRPC protocol field information

Field	Meaning	Space(bit)	Posistion
version	Version nimber	8	*Head*
reserved	Reserved for future	8	
magic	Magic number(0xb7d5)	16	
serial_id	Serial number	32	
total_len	Length of body	32	
timestamp	Timestamp	32	
serve_id	Server number ,1 for index, 2 for retrieval	8	*Body*
reserved	Reserved for future	8	
len	Length of data field	16	
data	Date of json format	variable	

Receiving data service uses asynchronous event-driven library called liboop, It bounds callback accept function to listening port and put connect sockets into the temporary buffer pool. Then read event is registered and the network data is constructed as a task request. Multiple working threads obtain a task from the pool of task and parse protocol, strip documents, write them into message queue.

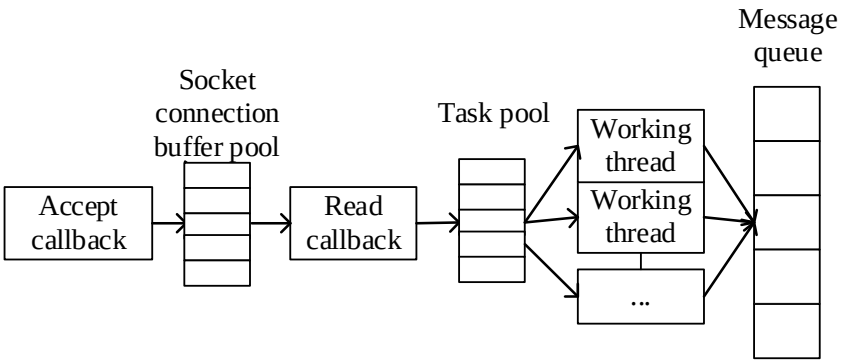

Fig. 2 data receiving service

Message Queue.Message queue uses POXIS shared memory mechanism, and establish a file in the /dev/shm directory. The files in this directory are not on disk, but entirely in memory. And then through the MMAP memory mapping [7], services can randomly access to the contents of the file, with high speed and flexibility of the process of sharing a larger resource. The queue references the design of Linux kernel data structure, kfifo circular queue. It uses lock free programming technology, namely reading and writing in single thread cases without the need for locks, while ensuring the thread security and the maximum needs of efficiency.

Indexing Service

The indexing service obtaines documents from the message queue through the following steps including establishing forward index, building the index item, inserting the inverted index to generate index structure. The index bases are divided into four levels, namely, a Level 1 temporary base, Level 2 temporary base, Level 1 persistent base and Level 2 persistent base. When the trigger condition is reached, the index base will start the merging process.

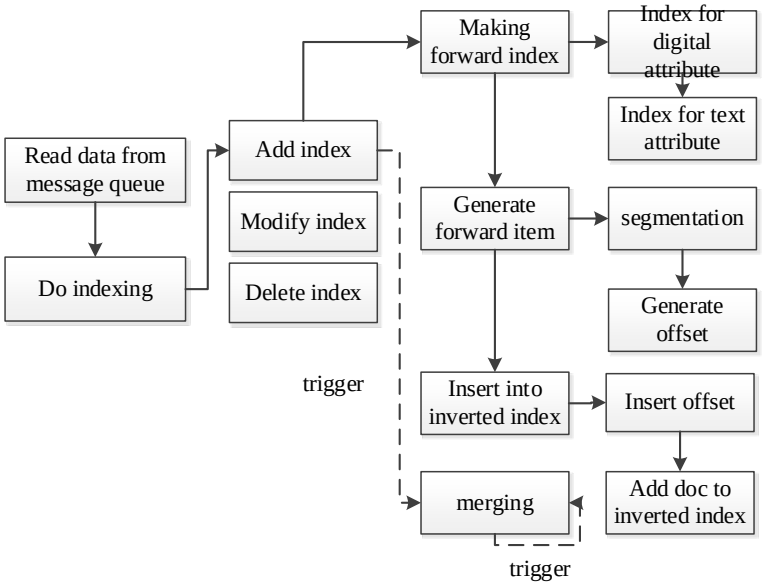

Fig. 3 indexing service

Structure of Index Base. Level 1 temporary base uses hash table structure to index the original documents. It is temporary and dynamic. It meets the requirement of real-time index which is the source of other index base and the basis of index merging. The following is index the data structure of Level 1 temporary base and Level 2 temporary base. Two temporary bases are permanently resident in internal memory. They are initialized by with a certain proportion of the allocation in determined System V shared memory.

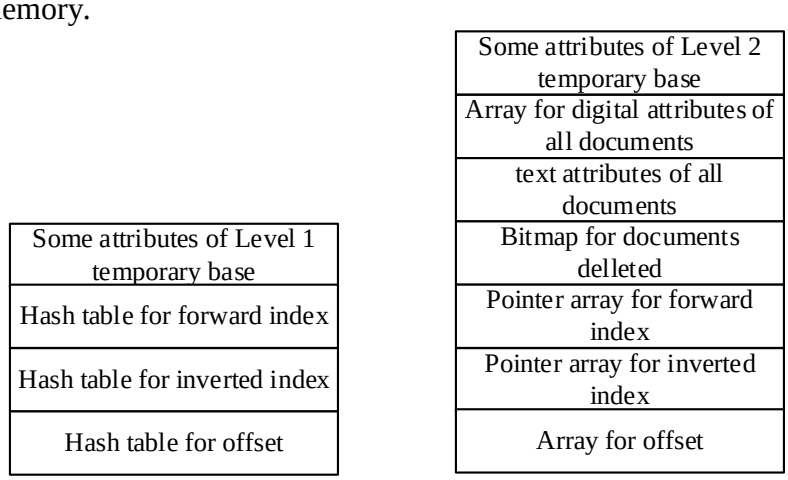

Fig. 4 Index structure diagram

Hash table for forward index is used to get forward index information according to external document ID. The document has internal and external ID, the external ID comes with document from input, usually is md5sum for the URL of the document. The internal ID is based on the sequence generated by the document sequence number started from 1. Internal ID will change with the merge of the index, which represents the default sort information of the document. Internal number plays an important role in index establishment and the connection of forward and inverted index.

The inverted index hash table is based on the term that all the documents contain, using separate chaining method to resolve conflicts. Each inverted term is composed of term attribute information and the internal document ID and offset ID of all documents containing the term. Offset ID points to the offset entity in the offset hash table. The Offset entity contains the field ID and the relative document start offset of the term in the document. Because of the shorter length text, If the offset is stored directly into the inverted index, it can not only make the inverted structure becoming longer, but also stored repeatedly in the inverted index. So Hecate only stores the ID of the offset in the inverted index and designs additional space for storage of offset. While reducing the burden of inverted index, it plays a role in removing duplicated offset. If there already exists an offset entity, it

can directly reuse this offset. By statistics, the repetition rate of offset is very high. The design not only saves the memory but also improves the efficiency of traversing the inverted index. The hash structure is used to satisfy the dynamic insertion of offset. The offset hash table contains a slot pointer which holds the ID of the first offset of the linear list and a pointer to the offset entity which holds first address of the stored offset. The offset is in a compact sequence. If there is a conflict with the offset entity, NextID is used to continue addressing. Each time there is a new offset, offset ID is added 1.

Level 2 temporary base is a temporary base from the merging of Level 1 temporary base. It is static and it does not require a hash structure, so we distribute the continuous space to store index. Level 2 temporary base also resides in the memory.

Digital attribute array contains forward index information of all numeric attributes of documents. The array's subscript is the internal ID of the document. The text attribute array holds all the text attribute information of Level 2 temporary base which stores consecutively to save space. Each storage of 1000*16KB is regarded as a page and it is immediately compressed when is written fully, while recording the address of document in memory before and after compression. While querying, according to the internal document ID we find the location of the page. After decompression, we use the address to locate text information.

The pointer array of forward index is sorted according to the external document ID. We can obtain the digital attribute and text attribute through the binary search of external document ID. Deleted-document bitmap is to record the internal ID of deleted documents. The inverted index array is an array of pointers sorted by term MD5. Term contains the offset ID is exactly the offset array index. Recalculating and sorting of the offset ID is needed while merging two index bases.

Level 1 persistent base comes from persistence of Level 2 temporary base and Level 2 persistent base comes from the merging of Level 1 persistent base. The purpose of designing the two level persistence base is to reduce the cost of the merging. Persistence base and Level 2 temporary base are of the same structure, but for some corresponding optimization according to the characteristics of a large amount of data and stability. The inverted index of term in the persistent base uses differential encoding for the internal ID of the document, which can save the storage space. Because the use of differential encoding, we can not use to binary search to find documents. So we need a jump table which includes real address. The text information and offset stored in the persistent base only record the relative offset to segment starting address, reducing the storage space of address.

Creation, Deletion and Modification of Index. Building index is only in Level 1 temporary base. Through getting data from the message queue, traversal of the document for the first time and analysis of the document structure, digital properties are established in the digital area and text information attributes are in the text area generation then forward index information is generated. And then travel the document for the second time to complete segmentation. The focus is that the formation of offset structure of the term is generated, then the offset information is inserted into the offset hash table structure in the inverted index of current term, if there is none, new term node will be established and term will be inserted at the end the inverted list. Now the establishment of a document in inverted index completes.

Short text has the characteristics of uneven distribution of words, so many terms are only contained by a small number of documents, or even single document, the design of a unified storage structure will result in a waste of memory. Moreover, the number of short text documents is much more than common that of the same volume. Even if the document storage is allocated dynamicly, memory utilization and allocation times need to be considered at the same time. Our system uses elastic structure of multiple types to solve the problem of uneven distribution of term. Default size of each document in term inverted index is 16 Byte and is enough to accommodate two documents. When the document number exceeds two, the space is converted into a structure that contains a head node of a single list structure. Term inverted will turn into a single list, each list node contains a number of arrays, each array can accommodate the document of a number of the 2^m, m is the distance to the tail of the list. The time complexity of searching is O (n log), n is the number of documents. Each element of an array is a structure that contains information about the internal ID of the document and its offset

ID. This design fully takes account the characteristics of short text into consider, such as good scalability, space saving, less system overhead. The average space utilization rate is 75%.
In terms of deletion of the index, if the data is in a Level 1 temporary base, directly delete the document in the forward index node; If in other bases, because it is static, we can only set 1 to the corresponding position in deleting bitmap next time when merging to complete really deletion. For modifying the index, if it is a numeric attribute modification, finding its index base to modify the digital properties; if it is a text attribute modification, delete the original document and reestablish.
Merging. Except for Level 1 temporary base, the other three bases are merged by the previous index bases. The index merging is always triggered. After a new document is entered or another merging process, if the index base size exceeds the threshold value, the merging will be triggered. When the index is merging, a backup index base is kept to ensure that the search service is provided to the outside. In order to ensure the consistency of data, new index request is not received when the index base is merging. The merging process contains several important steps:
For all the forward index document, it is sorted according to the default policy. Then generate new document internal ID and create a map of internal ID from current ID to previous ID of the document like (curid, hisid). Store the ordered new document according to new document ID in the new index base. At the same time, delete the documents in deleting bitmap. For the merge of Level 1 temporary base ,It need to be transferred from hash structure to sequential static structure, generates pointer array which points to forward index and is sorted according to the internal document ID.
Two inverted indies will merge when they have the same term. According to the map contains new and old document ID, update internal document ID in document list of the term. Offset information will be reorganized, and deduplicated. It is similar to regeneration of a new offset to the original base and new base. Regenerating an array of pointers to the inverted index of term sorted by MD5. For two persistent index base, they can be directly merged and not needed to resort the term in accordance with term's MD5.

Retrieval Service

The retrieval service receives external query, and makes grammar analysis, logical analysis and constructs query syntax tree structure. It queries cache and four index bases in parallel, merges results, processes logic *and*, *or*, *not*, scores the results, and returns results.
Dealing with Query. The core function is syntax analysis and the logical analysis for query. A complete query consists of several sub queries. Sub query can be text query, digital attribute query. Query support the syntax of the sort of specific attributes, the specified domain search, the specified domain aggregation, numerical calculation and comparison, and so forth.
We try to select feature words in retrieving. Feature words are subjective concepts, so we use importance of the attributes of the word to reflect the characteristics. The weight of named entities, such as companies, person's names, addresses, should be higher, which is automatically selected as the feature word. The proper nouns and some verbs have some weight. English and number of the query string have higher weight. The weight of dynamic bond of stop word[8] is small, the dynamic bond on non stop word will be more weighted, because bond words consist of information of both and words and single word. Because some query string is longe, considering the efficiency of retrieval, the number of feature selection is less than 3, and the number of total index words is limited to 7, which the core and non-core words are half and half. If the first query results in nothing, the service was downgraded to a maximum of only 2 core words.

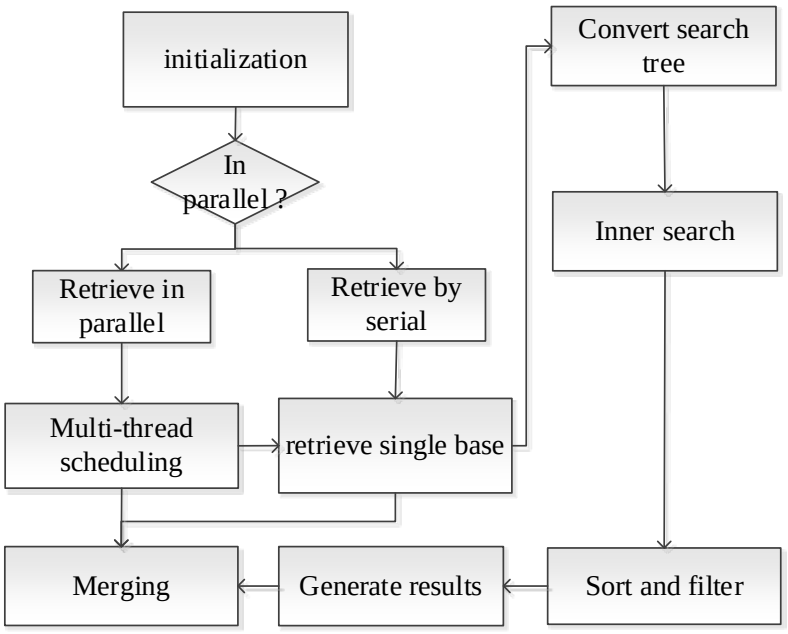

Fig. 5 flow chart of retrieval

Retrieval of Index Base. Retrieval initialization includes setting the number of AND truncation, sorting filter truncation number, sorting heap size, parallel search switch, the specified index base, etc.. Then decide to execute a serial search or parallel search. In parallel search, the search task is informed to each retrieval thread, and will be merged after all the results are completed. Each search thread still uses a serial retrieval.

Serial retrieval is based on the specified index base, the default query for all four index base. Retrieval for a single index base is completed by a function of a specified index base. Firstly cache is queried, it's key is made up by the index base number and query string signature. If the cache is not hit, then it retrieves the index base. The structure of the search tree is generated, and then the internal search function is called. After the search is completed, check whether it is needed to be put in the cache, and get its offset. Computing scores after some grammatical functions such as processing attribute filtering, attribute sorting, attribute aggregation. Using heap sort algorithm to sort the top K results from the forward index and get document information from forward index to fill the search results.

Internal search function contains a loop, checking out all from all the eligible document from term inverted list one by one, the input is non-leaf node in a query tree, which represents the relationship of AND OR NOT between term as well as the starting document ID. It returns the first document ID that is less than or equal to the initial document ID in accordance with the query tree condition, until it reached the query tree leaves. In the beginning or after the selection of a document, the nodes of inverted list must be sorted based on the maximum document ID of each list.

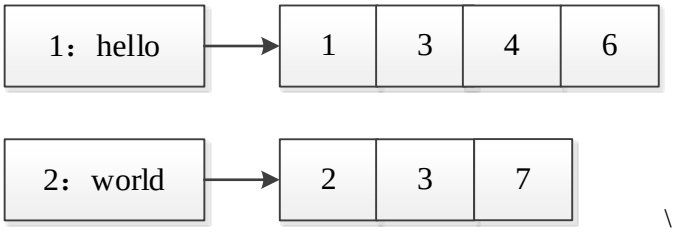

Fig. 6 Retrieval example

If you have the above inverted indexand want to search *hello world* for AND query: first of all, the reverted list of "hello" is check out. The detection ID is 6. We use 6 as the starting point for the detection of "world", 3 is selected. Then we use 3 for the starting point of the "hello" , and detect 3. So it return the document ID of 3 as result.

For OR query: first of all, for the "world" 7 is detected of 7, return 7; because 6 is greater than 3, so the next time we should begin from "hello".

For NOT query: 7 in the "world" was detected, it not the result. We set 6 as a starting point, 6 in the “hello” was detected, then to 5, in the two term are not detected, return 5.

$$P(Doc \mid Query) = \frac{P(Query \mid Doc)P(Doc)}{P(Query)} \quad (1)$$

Scoring and Sorting. Scoring is based on Bayes formula [9]. $P(Query \mid Doc)$ is query matching degree, $P(Doc)$ is the quality of the document score, because $P(Query)$ in the same search is the same, so the correlation score is the product of query matching degree and the quality of the document score.

$$P(Query \mid Doc) = \sum_i Score_i(Query, Doc) \quad (2)$$

The Query matching degree score is determined by each scorer, the formula and the following ideas are as follows:

a) A basic score called bm25_score is obtained according to BM25[10] algorithm

b) To calculate the text matching information score, we set coefficient respectively according to the nearest neighbor, the order, complete hit, part of the hit, long query.

c) Setting the full match punishment according to the number of content hits. If the full text matching is less than 3 times, then the matching factor is set to 0.5, the other is 1

d) When it hit title, score * = 1.1

$$P(Doc) = \prod_j Booster_j(Query, Doc) \quad (3)$$

The quality of the document is the product of boosters, the formula and the booster ideas are as follows:

a) Define different weight coefficients for different types of documents. For example, decreased in the following order, message, micro-blog, mail, news, comment.

b) Timeliness [11] factor, according to the time of the document to calculate the time damping.

b) Considering Language type factor to set language category coefficient, such as non-Chinese language, Japan and South Korea, traditional Chinese characters, simplified Chinese characters.

d) Page quality coefficient calculation. Currently it is calculated according to the length of the content with no other external information.

Performance Test

Environment of Experiment. Hecate is deployed on a RD450 ThinkServer server. It has a single memory of 32GB, 6-core Xeon E5-2609 V3 CPU, operating system of CentOS 6.5, hard disk of 1TB. It is equipped with 100Mbps Ethernet. The testing data used in the system is a collection of short text documents that are separated by the 400G paper document. Using the current popular open source search engine Elasticsearch (for shot ES) [12] as a contrast.

Test content includes online function test and offline function test. Offline function includes the time and space performance of index establishment. Online function test is focused on efficiency and accuracy of retrieval.

Index Performance . Using 100 million documents with the average length of around 60 Chinese characters to establish the index. The total amount of data indexed is around 20GB. The time performance between ES and Hecate of establishing the index is shown below. When the document number is less than 40 million, Hecate and ES have almost the same indexing time; when the document number reached 40 million to 60 million, the two systems have different advantages; when the document number is greater than 70 million, Hecate has obvious advantages over ES in terms of time performance, the average improvement is 13%, when the document number reaches 100 million It has the highest efficiency that is increased by 18%.

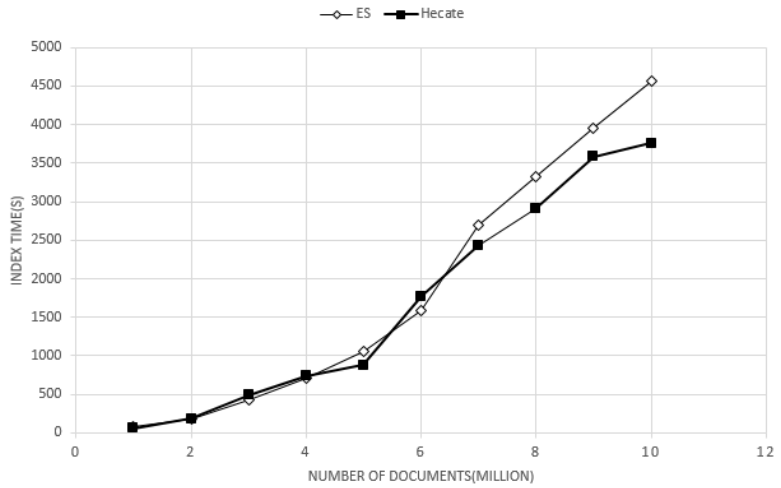

Fig. 7 comparison of Index time

In terms of the space performance of the index, because the Hecate has allocated a 10GB shared memory, the use of storage space is in the constant initially. When document number of ES is less than 50 million, It occupies less storage space When the document number is greater than 50 million, because the multistage structure index and Optimization for short text, space efficiency of Hecate is outstanding, while ES utilization rate is decreased. With 70 million documents, the enhance of the efficiency reaches its peak value. It only has the space of 71% occupied by ES. As for the space utilization rate, when the number of documents is largere Hecate has more advantages than ES.

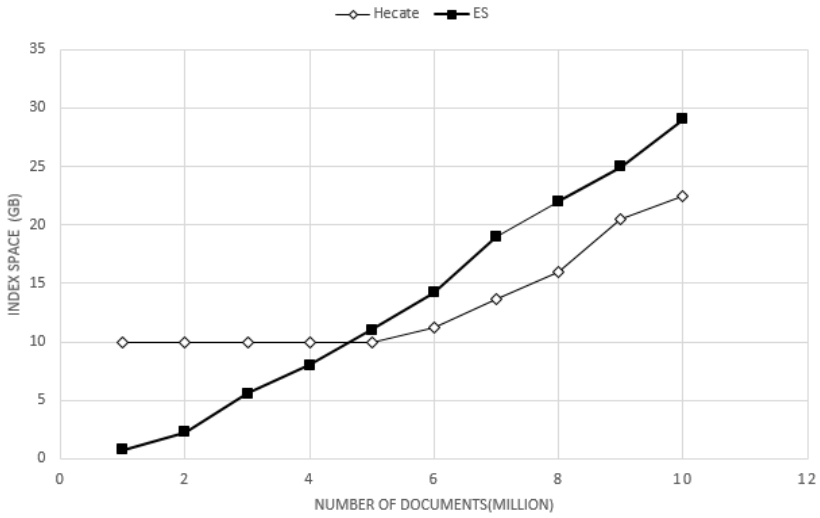

Fig. 8 comparison of index space

Retrieval Performance. In the test of retrieval time, choosing 10 commonly used search words, each word searches twice from 10 million documents to 100 million for 10 times. Testing index retrieval time performance when using cache or not respectively and recording average retrieval time of all words in the table below.

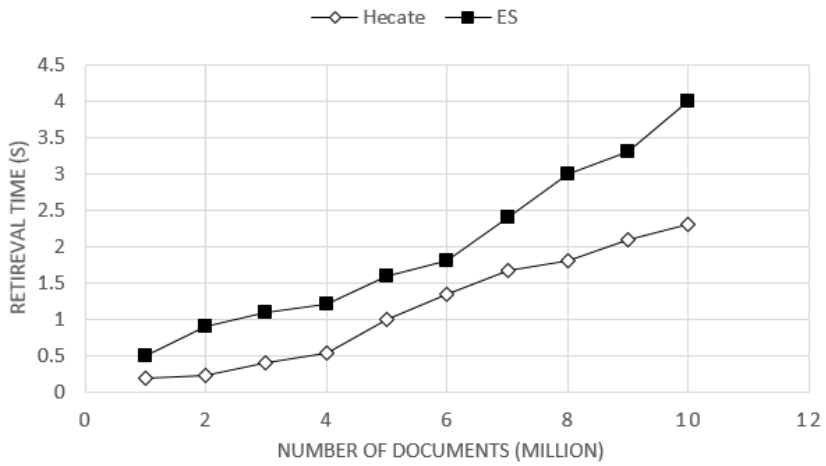

Fig. 9 comparison of retrieval time

From the figure above, due to the use of a large number of memory and cache, retrieval rate of Hecate is much higher than ElasticSearch. In the retrieval of 100 million document, Hecate use 2.3 seconds average, while the average time of ES is 4 seconds. The retrieval time is improved by 42%.

Sort of search results is also an important indicator of performance. Use of 1000 documents for test, for some of the key words we do sorting manually for the first n the documents, while the ES and Hecate index and retrieve them. An evaluation function is constructed, its purpose is measuring the average distance of all search results between documents labeled by people representing ideal ranking value and the actual ranking returned by search engine. Actual ranking $real(i)$ and ideal ranking $ideal(i)$ for a query Q, calculate the average score $AS(Average-Score)$ of the ranking results. From the following AS score table we can see, for short text, compared to ES, the accuracy of the retrieval is almost the same. Each has advantages in different number of documents' rankings.

$$AS(q)=\left(\sum_{j=1}^{n}\frac{1}{j}\right)\sum_{i=1}^{n}\left(\frac{1}{ideal(i)}\left(1-\frac{\left(ideal(i)-real(i)\right)^{2}}{n-1}\right)\right) \quad (4)$$

Table 2. Score of Hecate and ES

n	5	10	20	30	40
Hecate	0.93	0.904	0.83	0.81	0.8
ES	0.92	0.91	0.82	0.82	0.79

In summary, the performance of Hecate in the space of indexing, retrieval and indexing time and the efficiency are nearly the same as ES in small amount of data. When the index document number increases, performance is significantly better than Elasticsearch which is on behalf of the mainstream open source search engine. The accuracy performance of two system is nearly the same which means Hecate can meet standard of mainstream search engine.

Summary

The development of the Internet and the change of network information flow make short text becoming an important way of expression and the carrier of Internet information. The improvement of short text retrieval technology in search efficiency is of great significance to accelerate the dissemination of knowledge. In this paper, through the analysis of the characteristics of short text, we design and implement a short text oriented full-text retrieval system. It includes three aspects including document receiving service, index service and retrieval service. With optimizing the index structure of storage, retrieval process and selection of Chinese words, we implement an efficient full-text retrieval system for short text called Hecate. By comparing with the mainstream open source search engine Elasticsearch, it is proved that Hecate has certain advantages in time and space efficiency and its accuracy can reach the mainstream standard.

Acknowledgments

Associate Professor Yongping Xiong has been devoted to my thesis. My supervisor Professor Changqiao Xu also gives his precise advice on my paper. I really appreciate their work. I am indebted to my classmates who offer their own help. Finally I am also deeply grateful for the support of my parents. They give me strength.

References

[1] Zhang Xiaowei. Clustering algorithm and its application in the search engine system. Diss. Harbin University of Science and Technology, 2014.

[2] Ding Yuan. Sentiment analysis of Chinese short text. Diss. Beijing University of Posts and Telecommunications, 2015

[3] Peng Min et al. Massive short text clustering and topic extraction based on frequent item sets. Computer research and development 52.9 (2015): 1941-1953.

[4] Li Xuwei. Research on key technology of short text retrieval in micro-blog. Diss. Harbin Institute of Technology, 2013

[5] Gu Yanhui et al. A retrieval strategy based on short space text object. Journal of Peking University: Natural Science Edition 52.1 (2016): 120-126.

[6] Li Rui, and Wang Bin. "A micro-blog search model based on the author's modeling," Chinese Journal of information 28.2 (2014): 136-143.

[7] Zhang Yunjian et al. A user scenario reduction tool based on file memory mapping. Computer and modernization 1 (2016): 6-11.

[8] Wu Xiaona. Topic modeling algorithm based on feature, priori and constraint. Diss. Soochow University, 2014

[9] Wang Wei. "The inquiry teaching method of total probability formula and Bayes formula." science 28 (2015): Wenhui junkan 48-49.

[10] Zhao, Jiashu, J. X. Huang, and Z. Ye. "Modeling Term Associations for Probabilistic Information Retrieval." Acm Transactions on Information Systems 32.2(2014):1-47.

[11] Ondrusova, M., M. Psenkova, and V. Donath. "Reliable and Timely Event Notification for Publish/Subscribe Services Over the Internet."Networking IEEE/ACM Transactions on 22.1(2014):230-243.

[12] Singh, A., and H. Gonzalez Velez. "Hierarchical Multi-log Cloud-Based Search Engine." Eighth International Conference on Complex, Intelligent and Software Intensive Systems IEEE, 2014:211-219.