Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080139

A Two-Tier Distributed Full-Text Indexing System

Wei-Zhe Zhang[*], *Hui-Xiang Chen, Hui He and Gui Chen*

School of Computer Science and Technology, Harbin Institute of Technology, Harbin150001, China

Received: 22 Jun. 2013, Revised: 29 Oct. 2013, Accepted: 30 Oct. 2013
Published online: 1 Jan. 2014

Abstract: The performance of indexing systems is very important for a search engine. Usually, indexing systems on large-scale clusters can provide high search efficiency, but it brings expensive hardware costs. The costs would be greatly reduced if a distributed indexing system runs on small-scale clusters connected by the Internet. Two current inverted file partitioning schemes: document partitioning and term partitioning, have their merits individually. A two-tier distributed full-text indexing system is implemented, which uses document partitioning among the clusters and term partitioning inside each cluster. Our experiments show that the system performs well in search efficiency, resource consuming and load balance.

Keywords: Distributed indexing, document partitioning, term partitioning, search efficiency, load balance

1 Introduction

With the development of Internet, search engines have become essential tools for information retrieval from huge amounts of web Information Ocean [1]. Short response time and accurate search results are basic needs for users. However, large-scale clusters which have sufficient processing ability and adequate storage are needed to meet the requirements above. Commerce search engines can achieve this, but cause expensive hard-ware cost. The question is put forward from the idea that if a distributed search engine that runs on small-scale clusters connected by the Internet can be designed.

The structure of the index is very important for a search engine. Good index structure makes accurate and quick search. However, data on the Internet is dynamic, so index need to be updated to ensure accurate query. The easiest way of index updating is to rebuild the index, however, the web page set is so huge that it takes several days to finish and query requests are not available while rebuilding. If query requests were permitted using the old index, recall and precision of the search engine would not be ensured since the web set has changed. Nowadays, most search engines use incremental index updating strategy [2]. This paper is related to the Internet connection among clusters. Thus, burden of network transmission among clusters is a factor we need to consider.

Current two different distributed inverted file partitioning schemes: document partitioning and term partitioning [3]. The former is easier to implement, has higher parallelism and better load balancing. The latter is quicker in query routing, but worse in load balance. This paper introduces a two-tier distributed full-text indexing system, which combines advantages of document partitioning and term partitioningand provides good trade-off between search efficiency, resource consuming and load balance.

The rest of the paper is organized as follows. Section 2 introduces and compares the two current inverted file partitioning scheme for distributed index. Section 3 describes the network environment and implementation of the two-tier distributed full-text indexing system. Experimental results and analysis are shown in section 4 and finally, conclusion is presented in section 5.

2 Two Inverted File Partitioning Scheme for Distributed Index

An inverted index is an index data structure mainly storing a mapping from content, mainly words and also numbers, to its location in a database file, or a document, or a set of documents. All the words occurred in all web documents make up the vocabulary. The relationship of an inverted index can be represented using a matrix $(T \times D)$. $T = \{t_1, t_2, ..., t_n\}$ means vocabulary, and $D = \{d_1, d_2, ..., d_m\}$is the document set. $(T \times D)_{ij}$ is

[*] Corresponding author e-mail: wzzhang@hit.edu.cn

positive if word(i.e. term) t_i occurs in document d_j. The value of $(T \times D)_{ij}$ is decided by the frequency that t_i occurs in d_j, and also the frequency that t_i occurs in the whole document set.

To build distributed index[4,5], the matrix should be partitioned, and then each sub matrix is distributed to each index server. Currently, there are two partitioning schemes for distributed inverted files. Document partitioning: partitions D by document id horizontally. $D = \{D_1, D_2, ..., D_p\}$. Then each subset of D is distributed to each index server. Term partitioning: partitions T by word id vertically. $T = \{T_1, T_2, ...T_q\}$. Then each index server is in charge of each subset of T.

As for index building, a good partitioning method should be selected whether for document partitioning or term partitioning. The simplest way is to allocate each document or word randomly to each index server, which causes heavy network burden. For document partitioning, the index can be built by partitioning documents according to subjects. By this way, queries related to one subject will be forwarded directly to a specific index server. For term partitioning, the whole index is built on one server and then allocated to each index server. If web dataset is huge, large amounts of index files are transmitted on the Internet. Overall, document partitioning is easier to implement index building and has good expansibility because a new index server can be allocated to handle the index of new document. But term partitioning has not because the whole index needs to be rebuilt when the web document set is updated.

On the other hand, the two schemes are different in searching process. For document partitioning, each index server stores relation $T \times D_i$. A keyword in a query is possible to occur in any index server, so the query request of the word should be forwarded to all index servers. Then they all search in their index files and return search results to search servers. Therefore, searching based on document partitioning has good balance. However, it also brings high network burden and resource consuming. The more index servers are, the slower response time is due to larger network traffic and more time in returning results. For term partitioning, each index server stores relation $T_j \times D$. Therefore, the query request of a keyword will be directly forwarded to the specific index server that stores the word's inverted list, so network overhead and resource consuming is low. The drawback is bad load balance. Based on the discussion, term partitioning is superior to document partitioning in search efficiency.

3 Two-Tier Distributed Full-Text Indexing System

3.1 Network Environment

Our indexing system applies to small-scale clusters connected by the Internet. Inside each cluster, it is connected by high-speed LAN. Among the clusters, it is connected by the low-speed Internet, so the indexing system should minimize network transmissions among the clusters. See Figure 1.

Considering that network transmission among the clusters should be as small as possible, large data transmission can be confined inside the clusters, so term partitioning inside each cluster is a good choice. Besides, different clusters are connected by the low-speed Internet, using document partitioning ensures load balance among the clusters. Based on the discussion above, this paper proposes a two-tier partitioning scheme, which is using document partitioning among the clusters and term partitioning inside each cluster. And the two-tier distributed full-text indexing system is implemented using this scheme.

3.2 Implementation

D is the complete web document set and each cluster stores a subset of D, D_1, D_2, D_3, $D = \{D_1, D_2, D_3\}$. Document partitioning is used among the clusters. Each cluster build its local index files based on its web document set inde-pendently. Cluster1 builds index of D_1, Cluster2 builds index of D_2 and Cluster3 builds index of D_3. Inside each cluster, term partitioning is used. Take Cluster1 for example, it consists of n index servers.

Definition of symbols: $\forall t \in T, p(t)$ represents the frequency that t occurs in all queries. P is a specified threshold to decide whether t is a high frequency word or not. If $p(t) > p$, then t is added into PT, i.e. high frequency word set. $pt_1 \forall PT \wedge pt_2 \in PT, c(pt_1, pt_2)$ is the frequency that pt_1 and pt_2 both occur in the same query, i.e. How many queries contain both pt_1 and $pt_2.C$ is a specified threshold to determine whether pt_1 and pt_2 should be in the same group. In the following, the term partitioning algorithm inside Cluster1 is given.

Document partitioning among the clusters avoids large data transmission because each cluster builds its local index files. Moreover, term partitioning inside each cluster has good load balance, because low frequency words and the m groups of high frequency words are distributed averagely to index servers. Besides, partition of PT based on concurrence considers combination of high frequency words, thus improves search efficiency by reducing the relating index servers when a query is executed.

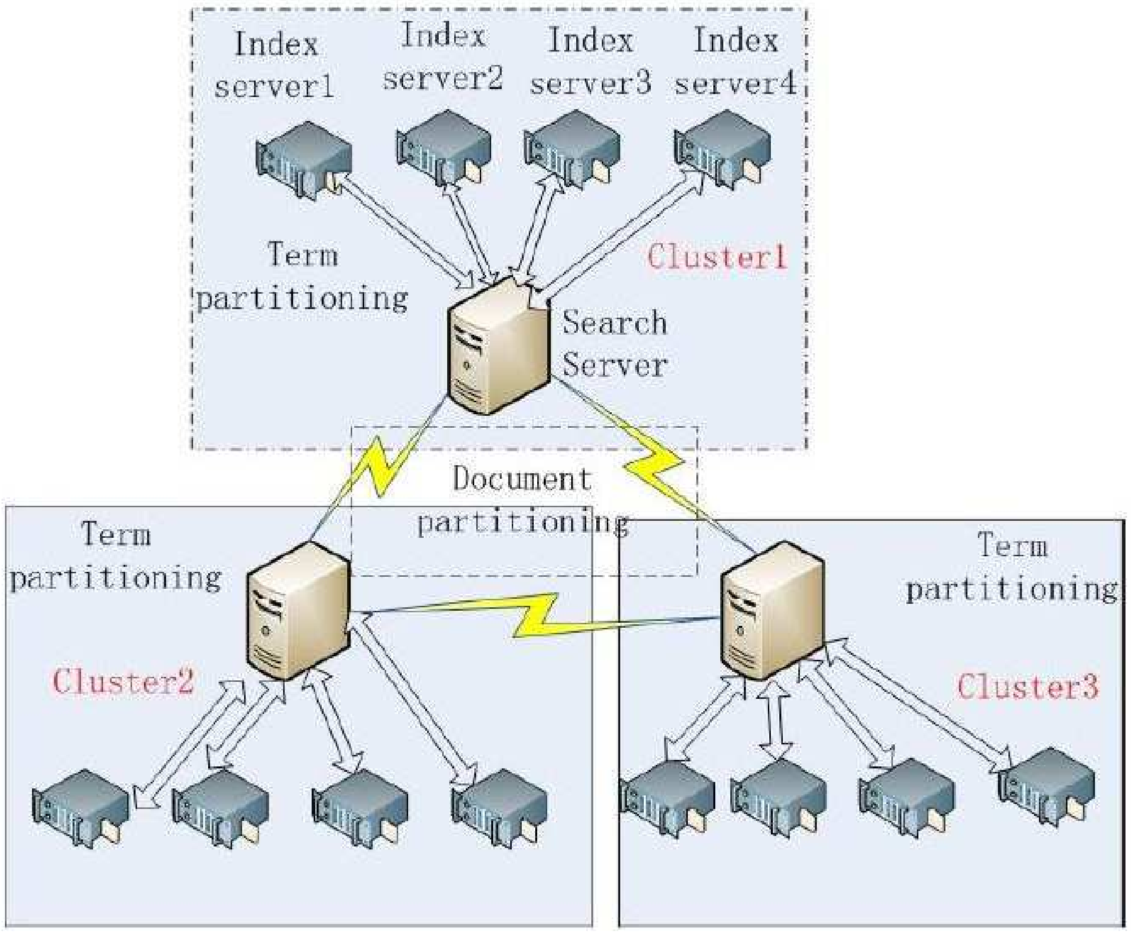

Fig. 1: Structure of the two-tier indexing system

```
Algorithm 1: Term Partitioning Algorithm in-side One Cluster (TPAOC)
1)sort vocabulary into two categories: high frequency words and low frequency words and allocate low frequency words to index servers
2) for each $t \in T$
3)   if $p(t) < p$ then
4)       randomly select an index server using consistent hash function
5)   else if $p(t) > p4$ then $PT = PT \cup \{t\}$
6) end for
7) partition words in $PT$ into $m$ groups based on co-occurrence
8) for each $(pt_a, pt_b) \in PT$
9)   if $c(pt_a, pt_b) > C$ then
10)      if $pt_a$ and $pt_b$ are not in any group
11)          then create a new group $c_i$
12)              $p(c_i) = max\{p(pt_a), (pt_b)\}$
13)          else if $pt_a$ is in group $c_j$ and $pt_b$ is not in any group before
14)              then add $pt_b$ into $c_j$
15)          $p(c_j) = max\{p(c_j), p(pt_b)\}$
16)      else if $pt_a$ is not in any group before and $pt_b$ is in group $c_j$
17)          then add $pt_a$ into $c_j$
18)          $p(c_j) = max\{p(c_a), p(pt_j)\}$
19)          else if $pt_a$ is in group $c_i$ and $pt_b$ is in group $c_j$
20)          then merge $c_i$ and $c_j$ into $c_i$
21)          $p(c_i) = max\{p(c_i), p(c_j)\}$
22)          end if
23)      end if
24) end for
25) //assign these m groups of words to n index servers
26) distribute $PT = c_1, c_2, c_3 ... c_m$ to $n$ index servers averagely by $p(c_i)$
```

3.3 Search Process

There are four kinds of roles in this system: Centralized search engine, cluster search engine, index server and

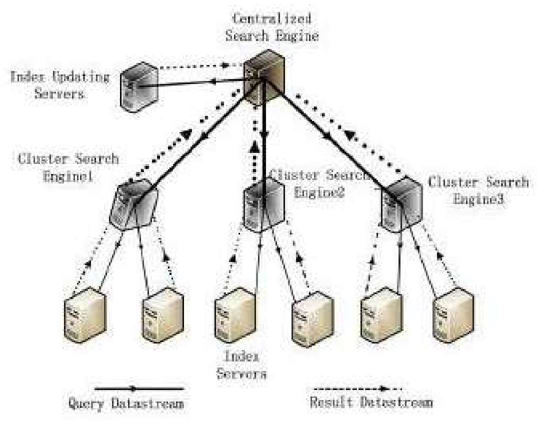

Fig. 2: Architecture of the two-tier distributed full-text indexing system

index updating server.

When a user sends query request, the search process in the system is as follows: first, the cen-tralized search engine receives the query request, and then forwards the request to all cluster search engines; second, each cluster search engine receives the request, and segments the query into keywords. Then it forwards each keyword to its corresponding index server, based on the term partitioning scheme and addressing of consistent hash; then each index server receives its keywords, and search in its inverted list to get results. Then each returns searching results to its cluster search engine; later each cluster search engine sorts all the searching results received and return sorted results to the centralized search engine; finally, the centralized search engine merges and passes the final results to the user.

The index updating server solves the complexity of rebuilding the whole index inside each cluster caused by dynamic web document set. It ensures normal query service even if the web document set has been updated. The whole index is rebuilt periodically or when the index updating server is fully loaded.

4 Experimental Results and Analysis

We use three computers(Configuration: Intel(R) Xeon(TM) 3.20GHz Quad-Core CPU, 280G HDD, 4G Memory) to simulate three clusters, and another computer(Configuration: Intel(R) Xeon Duo-Core CPU, 4G Memory) to represent centralized search engine. Then set the bandwidth among the clusters 2Mbs, and bandwidth inside each cluster 100Mbs. Assuming that the three clusters have 6, 8, 6 numbers of index servers respectively, and each has a cluster search engine. In this simulation environment, we implement three systems for the document partitioning and term partitioning. We use the full version of query log of users released by Sougou Lab(i.e. SougouQ) as input query set, internet thesaurus of Sougou Lab(SougouW) as the main vocabulary, randomly select 500,000,000 webs after content extraction from over 1 billion webs in internet corpus(SougouT) as the test document set. After segmented into keywords using IKAnalyz-er(http:code.google.compik-analyzer),each query contains three keywords on average(see Table 1).

$p(t)$ is calculated by making statistics of the number of queries each keyword appears in. PT consists of the top 3000 words which have higher $p(t)$. Then PT is partitioned using TPAOC algorithm. Later, we select 5 groups of queries (each group has 5000 queries) based on the distribution in Table 1. The 5 groups of queries are input serially into the centralized search engine. We record total response time for each group, then the throughput of each index system is got: $throughput = total$ response time5000. See Table 2.

For each query, calculate the amount of index servers which receive query request in the search process, we call this value QM $\overline{QM}$ is the mean value of QM for one group of queries. $\overline{QM}$ of the 5 groups of queries are as follows. In Table 2, the mean value of TDIS is the 66.5, which is higher than 64.3 and 47.0. As group varies, throughput changes only a little. And except in group 2(TDIS 60.7, DP 65.3), the throughput of TDIS is always the highest. Therefore, we conclude that TDIS do best in

Table 1: Distribution of number of keywords each query contains

How many keywords each query contains on average	Percentage
1	12.88%
2	28.91%
3	26.39%
4	15.40%
5	8.15%
>5	8.27%

Table 2: Experiment results of throughput for three systems

Group of queries \ System type	Two-tier distributed indexing system(qps)	Document partitio-ning(qps)	Term partitio-ning(qps)
1	67.3	64.9	48.4
2	60.7	65.3	49.5
3	73.5	66.1	47.4
4	61.4	59.3	44.1
5	69.8	65.7	45.2
mean value	66.5	64.3	47.0

Table 3: $\overline{QM}$ of the 5 groups of queries

Group of queries \ System type	TDIS(qps)	DP(qps)	TP(qps)
1	4.69	20	1.67
2	4.31	20	1.58
3	5.35	20	2.31
4	4.37	20	1.73
5	4.25	20	1.59
mean value	4.80	20	1.78

throughput, which indicates the highest search efficiency. In Table 3, the mean value of TDIS is 4.80, which is not much bigger than 1.78, but much smaller than 20. As group differs, the fluctuation of $\overline{QM}$ is small. $\overline{QM}$ of TDIS always not much bigger than TP but much smaller than DP. Thus, TDIS behaves well in resource con-suming (i.e. low resource consuming).

Moreover, we make statistics of load balance of each index server, which is represented by LBI: $LBI{=}the\ amount\ of\ queries\ that\ one\ index\ server\ has\ processed\ the\ amount\ of\ queries\ in\ the\ whole\ query\ set$. Then, for each partitioning scheme, we calculate the coefficient of variance (CV)based on the $LBIs$ of all twenty index servers, as is shown in Table 4.

Table 4: CV of twenty index servers in load balancing for 3 systems

System type in load balancing	CV of all twenty index servers
TDIS	8.80
DP	0
TP	22.32

5 Conclusions

This paper proposes a two-tier partitioning scheme combining the strengths of document partitioning and term partitioning, and implement the two-tier distributed full-text indexing system based on this. Experiment results show that the system has behaves well in search efficiency and resource consuming, load balance. Efficiency of our system will be greatly improved if caching and parallelization are used. Our future research will focus on the aspect.

Acknowledgement

This work was supported in part by NSFC under Grant Nos. 61173145, National High Technology Research and Development Program of China under Grant No. 2011AA010705, National Basic Research Program of China under Grant No.G2011CB302605.
The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] GlebSkobeltsyny, ToanLuuy, etc. Web Text Retrieval with a P2P Query-Driven Index. Amsterdam:30th annual international ACM SIGIR conference on Research and development in information retrieval, 686 (2007).
[2] Nicholas Lester, Alistair Moffat, Justin Zobel. Fast On-Line Index Construction by Geometric Partitioning. Ap-plications of Digital Information and Web Technologies, 90-95 (2008).
[3] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley-Longman, (1999).
[4] Melnik, S. Raghavan, S. Yang, B. Garcia-Molina, H. Building a Distributed Full-Text Index for the Web. ACM ASSOCIATION FOR COMPUTING MACHINERY, **19**, 217-241 (2001).
[5] Marco Hentschel, Maozhen Li, Mahesh Ponraj, etc. Distributed Indexing for Resource Discovery in P2P Networks. ShangHai: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009,Volume 00550–555M. Stojanovic, Low complexity OFDM detector for underwateracoustic channels. Proceedings of the IEEE Oceans Conference, 1-6 (2006).

Wei-Zhe Zhang received the B.S., M.S. and Ph.D. degree in computer science from Harbin Institute of Technology, Harbin, China. Since August 2003, he has been with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China, where he became a Professor in December 2012. His research interests include network computing, parallel computing. He is the corresponding author of this paper. His email is wzzhang@hit.edu.cn

Hui-Xiang Chen was born in He Nan in 1989. He received the B.S. degree from the Department of Computer and Communication Engineering, Jilin University, Jilin, China, in 2011. He is currently M.S. student of the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China, in 2011. His research interests include parallel computing, cloud computing

Hui He received the B.S., M.S. and Ph.D. degree in computer science from Harbin Institute of Technology, Harbin, China. Since September 1999, she has been with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China, where she became an Associate Professor in October 2007. Her research interests include network computing, network security

Gui Chen now studies the Bachelor degree at School of Software in Harbin Institute of Technology, China. His research interests include parallel and distributed system, cloud computing.