Effect of Inverted Index Partitioning Schemes on Performance of Query Processing in Parallel Text Retrieval Systems*

B. Barla Cambazoglu[1], Aytul Catal[2], and Cevdet Aykanat[1]

[1] Bilkent University, Department of Computer Engineering
06800 Bilkent, Ankara, Turkey
{berkant, aykanat}@cs.bilkent.edu.tr
[2] Scientific and Technological Research Council of Turkey (TÜBİTAK)
06100 Kavaklıdere, Ankara, Turkey
aytul.catal@iltaren.tubitak.gov.tr

Abstract. Shared-nothing, parallel text retrieval systems require an inverted index, representing a document collection, to be partitioned among a number of processors. In general, the index can be partitioned based on either the terms or documents in the collection, and the way the partitioning is done greatly affects the query processing performance of the parallel system. In this work, we investigate the effect of these two index partitioning schemes on query processing. We conduct experiments on a 32-node PC cluster, considering the case where index is completely stored in disk. Performance results are reported for a large (30 GB) document collection using an MPI-based parallel query processing implementation.

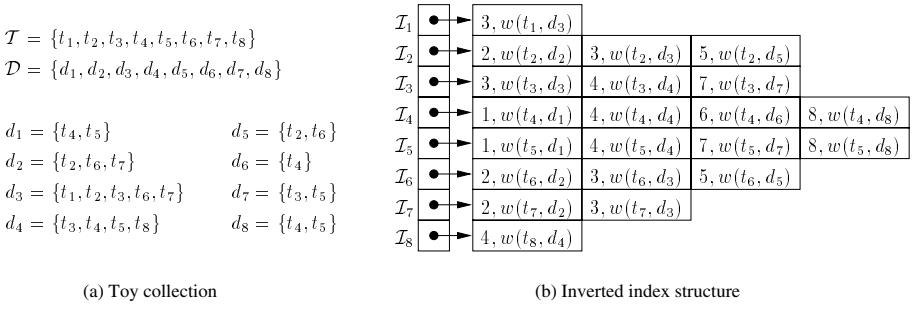
1 Introduction

The basic duty of a text retrieval system is to process user queries and present the users a set of documents relevant to their queries [1]. For small document collections, processing of a query can be performed over the original collection via full text search. However, for efficient query processing over large collections, an intermediate representation of the collection (i.e., and indexing mechanism) is required. Until the early 90's signature files and suffix arrays were the choice of most text retrieval system designers [2]. In the last decade, inverted index data structure [3,4] replaced these popular structures and currently appears to be the only choice for indexing large document collections.

An inverted index is composed of a set of inverted lists $\mathcal{L} = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_T\}$, where $T = |\mathcal{T}|$ is the size of the vocabulary $\mathcal{T}$ of the indexed document collection $\mathcal{D}$, and an index pointing to the heads of the inverted lists. The index part is usually small enough to fit into the main memory, but inverted lists are stored on the disk. Each list $\mathcal{I}_i \in \mathcal{L}$ is associated with a term $t_i \in \mathcal{T}$. An inverted list contains entries (called postings) for the documents containing the term it is associated

* This work is partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under project EEEAG-106E069.

A. Levi et al. (Eds.): ISCIS 2006, LNCS 4263, pp. 717–725, 2006.



(a) Toy collection (b) Inverted index structure

Fig. 1. The toy document collection used throughout the paper

with. A posting $p \in \mathcal{I}_i$ consists of a document id field $p.d = j$ and a weight field $p.w = w(t_i, d_j)$ for a document d_j in which term t_i appears. $w(t_i, d_j)$ is a weight which shows the degree of relevance between t_i and d_j using some metric.

Fig. 1-a shows the document collection that we will use throughout the examples in the paper. This toy document collection $\mathcal{D}$ contains $D = 8$ documents, and its vocabulary $\mathcal{T}$ has $T = 8$ terms. There are $P = 21$ posting entries, in the set $\mathcal{P}$ of postings. Fig. 1-b shows the inverted index built for this document collection.

2 Parallel Text Retrieval

In practice, parallel text retrieval architectures can be classified as: inter-query-parallel and intra-query-parallel architectures. In the first type, each processor in the parallel system works as a separate and independent query processor. Incoming user queries are directed to client query processors on a demand-driven basis. Processing of each query is handled solely by a single processor. Intra-query-parallel architectures are typically composed of a single central broker and a number of client processors, each running an index server responsible from a portion of the inverted index. In this architecture, the central broker redirects an incoming query to all client query processors in the system. All processors collaborate in processing of the query and compute partial answer sets of documents. The partial answer sets produced by the client query processors are merged at the central broker into a final answer set, as a final step.

In general, inter-query-parallel architectures obtain better throughput while intra-query-parallel architectures are better at reducing query response times. Further advantages and disadvantages and a brief comparison are provided in [5]. In this work, our focus is on intra-query-parallel text retrieval systems on shared-nothing parallel architectures.

3 Inverted Index Partitioning

In a K-processor, shared-nothing, intra-query-parallel text retrieval system, the inverted index is partitioned among K index servers. The partitioning should be

performed taking the storage load of index servers into consideration. If there are $|\mathcal{P}|$ posting entries in the inverted index, each index server S_j in the set $\mathcal{S}=\{S_1, S_2, \ldots, S_K\}$ of index servers should keep an approximately equal amount of posting entries as shown by

$$\text{SLoad}(S_j) \simeq \frac{|\mathcal{P}|}{K}, \qquad \text{for } 1 \leq j \leq K, \tag{1}$$

where $\text{SLoad}(S_j)$ is the storage load of index server S_j. The storage imbalance should be kept under a satisfactory value.

In general, partitioning of the inverted index can be performed in two different ways: term-based or document-based partitioning. In the term-based partitioning approach, each index server S_j locally keeps a subset $\mathcal{L}_j^{\text{t}}$ of the set $\mathcal{L}$ of all inverted lists, where

$$\mathcal{L}_1^{\text{t}} \cup \mathcal{L}_2^{\text{t}} \cup \ldots \cup \mathcal{L}_K^{\text{t}} = \mathcal{L}, \text{ and} \tag{2}$$

$$\mathcal{L}_i^{\text{t}} \cap \mathcal{L}_j^{\text{t}} = \emptyset, \qquad \text{for } 1 \leq i, j \leq K, i \neq j. \tag{3}$$

In this technique, all processors are responsible for processing their own set of terms, that is, inverted lists are assigned to index servers as a whole. If an inverted list $\mathcal{I}_i$ is assigned to index server S_j (i.e., $\mathcal{I}_{ji}^{\text{t}}=\mathcal{I}_i$), any index server $\mathcal{S}_k$ other than $\mathcal{S}_j$ has $\mathcal{I}_{ki}^{\text{t}}=\emptyset$.

Alternatively, the partitioning can be based on documents. In the document-based partitioning approach, each processor is responsible for a different set of documents, and an index server stores only the postings that contain the document ids assigned to it. Each index server S_j keeps a set $\mathcal{L}_j^{\text{d}}=\{\mathcal{I}_{j1}, \mathcal{I}_{j2}, \ldots, \mathcal{I}_{jT}\}$ of inverted lists containing subsets $\mathcal{I}_{ji}^{\text{d}}$ of every inverted list $\mathcal{I}_i \in \mathcal{L}$, where

$$\mathcal{I}_{1i}^{\text{d}} \cup \mathcal{I}_{2i}^{\text{d}} \cup \ldots \cup \mathcal{I}_{Ki}^{\text{d}} = \mathcal{I}_i, \qquad \text{for } 1 \leq i \leq T, \text{ and} \tag{4}$$

$$\mathcal{I}_{ji}^{\text{d}} \cap \mathcal{I}_{ki}^{\text{d}} = \emptyset, \qquad \text{for } 1 \leq j, k \leq K, j \neq k, 1 \leq i \leq T, \tag{5}$$

and it is possible to have $\mathcal{I}_{ji}^{\text{d}}=\emptyset$.

In Fig. 2-a and Fig. 2-b, the term- and document-based partitioning strategies are illustrated on our toy document collection for a 3-processor parallel system. The approach followed in this example is to assign the postings to processor in a round-robin fashion according to term and document ids. This technique is used in [6].

4 Previous Work

There are a number of papers on the inverted index partitioning problem in parallel text retrieval systems. We briefly overview three relevant publications.

Tomasic and Garcia-Molina [6] examine four different techniques to partition an inverted index on a shared-nothing distributed system for different hardware configurations. The system and disk organizations described in this paper correspond to the term- and document-based partitioning schemes we previously

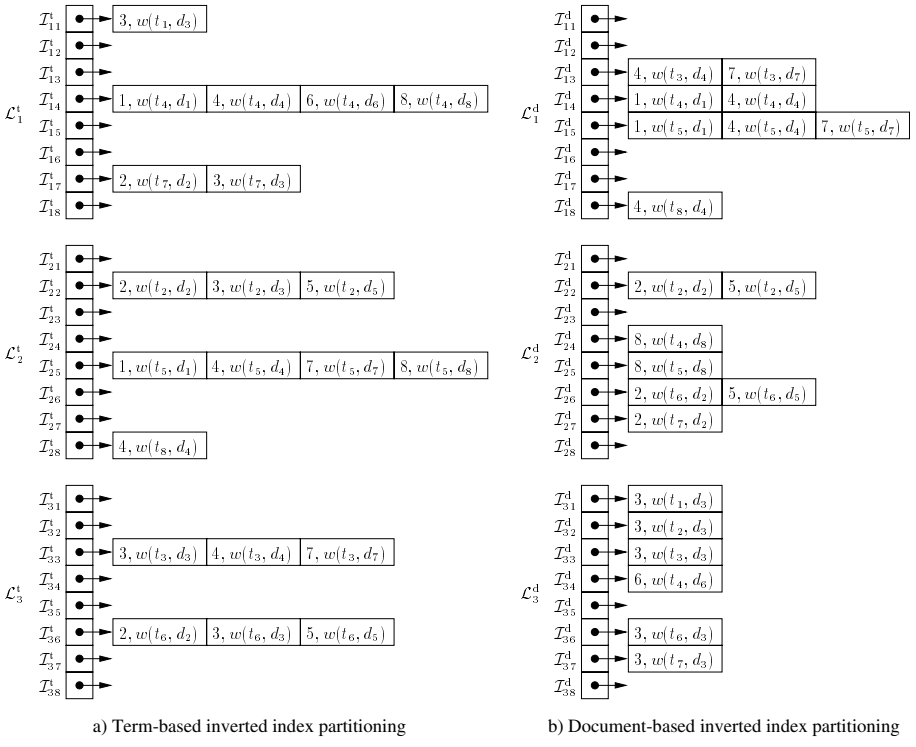


Fig. 2. 3-way term- and document-based partitions for our toy inverted index

described, respectively. The authors verify the performance of the techniques by simulation over a synthetic dataset and use the boolean model for similarity calculations between documents and queries. Their results indicate that document-based partitioning performs well for long documents, whereas term-based partitioning is better on short-document collections.

Jeong and Omiecinski [7] investigate the performance of the two partitioning schemes for a shared-everything multiprocessor system with multiple disks. As in [6], they use the boolean ranking model and work on synthetic datasets. They conduct experiments especially on term skewness. For term-based partitioning, they propose two heuristics for load balancing. In their first heuristic, they partition the posting file with equal posting sizes instead of equal number of terms. In their second heuristic, they consider the term frequencies as well as posting sizes. The results of their simulation show that term-based partitioning is better when term distribution is less skewed in the document collection, and document-based partitioning should be preferred otherwise.

Baeza-Yates and Ribeiro-Neto [8] apply the two partitioning schemes on a shared-nothing parallel system. In their work, they refer to term- and document-based partitioning schemes as global and local index organizations, respectively. For document ranking, they use the vector-space model and conduct their experiments on a real-life document collection. Their results show that term-based

Table 1. A comparison of the previous works on inverted index partitioning

Authors	Tomasic and Garcia-Molina	Jeong and Omiecinski	Riberio-Neto and Baeza-Yates
Year	1993	1995	1999
Target architecture	shared-nothing parallel	multi-disk PC	shared-nothing parallel
Ranking model	boolean	boolean	vector-space
Partitioning model	round-robin	load-balanced	load-balanced
Dataset	synthetic	synthetic	real-life

partitioning performs better than document-based partitioning in the presence of fast communication channels. Table 1 summarizes and compares the above-mentioned works on inverted index partitioning.

All performance results presented so far are based on simulations. In this work, we investigate the effect of the two partitioning schemes using an actual, MPI-based, experimental parallel text retrieval system, Skynet[1], implemented on a 32-node PC cluster.

5 Parallel Query Processing

Processing of a query in a parallel text retrieval system follows several steps. These steps slightly differ depending on whether term-based or document-based inverted index partitioning schemes are employed. In term-based partitioning, since the whole responsibility of a query term is assigned to a single processor, the central broker splits a user query $\mathcal{Q} = \{t_{q_1}, t_{q_2}, \ldots, t_{q_Q}\}$ into K subqueries. Each subquery $\mathcal{Q}_i$ contains the query terms whose responsibility is assigned to index server S_i, that is, $\mathcal{Q}_i = \{q_j : t_{q_j} \in \mathcal{Q} \text{ and } \mathcal{I}_{q_j} \in \mathcal{L}_i^t\}$. Then, the central broker sends the subqueries over the network to the index servers. Depending on the query content, it is possible to have $\mathcal{Q}_i = \emptyset$, and in that case, no subquery is sent to index server S_i. In document-based partitioning, postings of a term are distributed on many processors. Hence, unless a $K \times T$-bit term-to-processor mapping is stored in the central broker, each index server is sent a copy of the original query, that is, $\mathcal{Q}_i = \mathcal{Q}$.

Once an index server receives a subquery, it immediately accesses its disk and reads the inverted lists associated with the terms in the subquery. For each query term $t_{q_j} \in \mathcal{Q}_i$, inverted list $\mathcal{I}_j$ is fetched from the disk. The weight $p.w$ of each posting $p \in \mathcal{I}_j$ is used to update the corresponding score accumulator for document $p.d$. When all inverted lists are read and accumulator updates are completed, index server S_i transfers the accumulator entries (document ids and scores) to the central broker over the network, forming a partial answer set $\mathcal{A}_i$ for query $\mathcal{Q}$.

During this period, the central broker may be busy with directing other queries to index servers. For the final answer set to the query to be generated, all partial

[1] Skynet search engine: available at `http://skynet.cs.bilkent.edu.tr`

answer sets related with the query must be gathered at the central broker. The central broker merges the received K partial answer sets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_K$ and returns the most relevant (highly-ranked) document ids as the complete answer set to the user submitted query $\mathcal{Q}$.

In term-based partitioning, accessing a term's inverted list requires a single disk access, but reading the list (i.e., posting I/O) may take a long time since the whole list is stored at a single processor. Similarly, the partial answer sets transmitted by the index servers are long. Hence, the overhead of term-based partitioning is mainly at the network, during the communication of partial answer sets. Especially, in cases where the partial answer sets are long or inverted lists keep additional information such as information on term positions, this communication overhead becomes a bottleneck.

In document-based partitioning, disk accesses are the dominating overhead in total query processing time, especially in the presence of slow disks and a fast network. $O(K)$ disk seeks are required in the worst case to read the inverted list of a term since the complete list is distributed at many processors. However, the inverted lists retrieved from the disk are shorter in length, and hence posting I/O is faster. Moreover, in case the user is interested in only the top s documents, no more than s accumulator entries need to be communicated over the network (no document with a rank of $s+1$ in a partial answer set can take place among the top s documents in the global ranking).

6 Experiments

6.1 Setting

The hardware platform used in the experiments is a 32-node PC cluster interconnected by a Gigabit Ethernet switch. Each node contains an Intel Pentium IV 3.0 GHz processor, 1 GB of RAM, and runs Mandrake Linux, version 10.1. The sequential query processing algorithm is a term-ordered algorithm with static accumulator allocation [9]. The parallel query processing code is implemented in C using the LAM/MPI [10] library.

As the document collection, results of a large crawl performed over the '.edu' domain (i.e., the educational US Web sites) is used. The entire collection is 30 GB and contains 1,883,037 Web pages. After cleansing and stop-word elimination, 3,325,075 distinct index terms remain. The size of the inverted index constructed using this collection is around 2.7 GB. In term-based (document-based) partitioning, terms (documents) are alphabetically sorted and assigned to K index servers in a round-robin fashion using the distribution scheme of [6].

6.2 Results

Table 2 displays the storage imbalance in terms of the number of postings and inverted lists for the two partitioning schemes with varying number of index servers, $K = 2, 8, 32$. This table also shows the total number of disk accesses, the total volume of disk I/O, and the total volume of communication as well

Table 2. Performance comparison of the term- and document-based partitioning

	term-based			document-based		
	K=2	K=8	K=32	K=2	K=8	K=32
imbal. in posting storage (%)	0.7	6.9	17.5	0.1	0.2	0.7
imbal. in inverted list storage (%)	0.0	0.0	0.0	0.8	1.4	3.2
number of disk accesses	272	272	272	543	2161	8619
imbal. in disk accesses (%)	2.9	20.6	64.7	0.2	0.7	0.2
total volume of I/O (MB)	38.6	38.6	38.6	38.6	38.6	38.6
imbal. in I/O (%)	7.4	38.5	123.7	0.0	0.1	0.5
total comm. volume (MB)	36.1	38.0	38.5	33.3	33.3	33.3
imbal. in comm. volume (%)	7.4	38.5	123.7	0.0	0.1	0.5

as the respective imbalances observed in processing a set of 100 queries (having 1 to 5 terms) over the parallel text retrieval system. As expected, the number of disk accesses linearly increases with increasing number of index servers for document-based partitioning and is constant for term-based partitioning. However, the term-based scheme suffers from a considerable load imbalance in disk accesses as the number of index servers increases, i.e., some index servers perform quite more disk accesses than the others. The total volume of communication for transmitting PASs from index servers to the central broker is slightly higher for the case of term-based partitioning. Also, high imbalance rates are observed in posting I/O and hence PAS communication in this type of partitioning.

Fig. 3 shows the query processing performance with increasing number of query terms for different partitioning techniques and numbers of index servers. In this experiment, the central broker submits a single query to the index server and waits for completion of the answer set before submitting the next query. According to the figure, document-based partitioning leads to better response times compared to term-ordered partitioning. This is due to the more balanced distribution of the query processing load on index servers in the case of document-based partitioning. The results show that term-based partitioning is not appropriate for text retrieval systems, where queries arrive to the system infrequently. The poor performance of term-based partitioning is due to the imbalance in the number of disk accesses as well as communication volumes of index servers.

Fig. 4 presents the performance of the system with batch query processing. In these experiments, a batch of 100 queries, each containing between 1 and 5 query terms, was submitted to the system at the same time. The results indicate that term-based partitioning results in better throughput, especially as the number of index servers increases. This is mainly due to the better utilization of index servers and the capability to concurrently process query terms belonging to different queries. For document-based partitioning case, the number of disk accesses becomes a dominating overhead. In our case, after 8 index servers, the throughput starts to decrease.

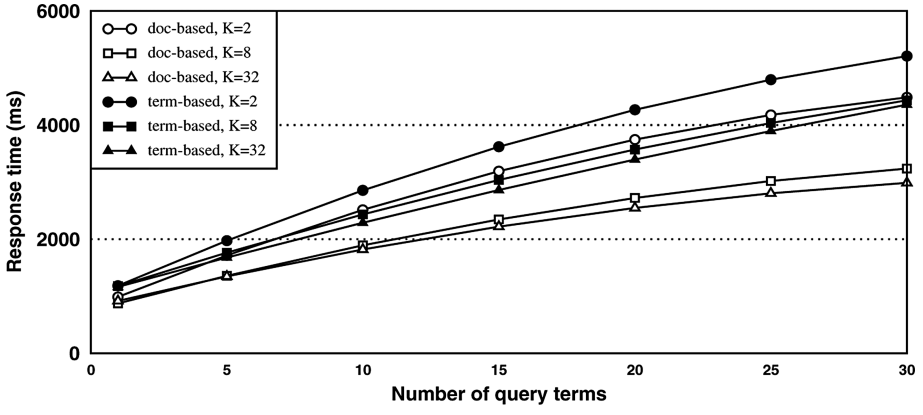


Fig. 3. Response times for varying number of query terms

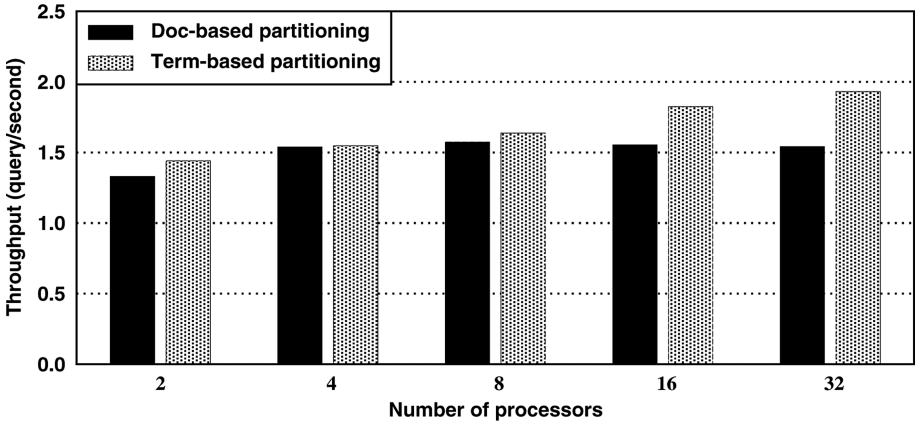


Fig. 4. Throughput with varying number of index servers

7 Conclusion

We have conducted experiments to illustrate the performance of two inverted index partitioning techniques on a recently-built, 32-node PC cluster system. We have implemented a parallel text retrieval system capable of working with both document-based and term-based partitioning schemes. We have conducted experiments to evaluate the response times and throughput of an MPI-based parallel query processing implementation. The results indicate that, for batch query processing, term-ordered partitioning produces superior throughput. However, for the case where queries are infrequently submitted, document-based partitioning should be preferred.

References

1. L. Page, S. Brin, The anatomy of a large-scale hypertextual web search engine. In: Proceedings of the Seventh World-Wide Web Conference. (1998) 107–117
2. Croft, W.B., Savino, P.: Implementing ranking strategies using text signatures. ACM Transactions on Office Information Systems **6**(1) (1988) 42–62
3. Zobel, J., Moffat, A., Sacks-Davis, R.: An efficient indexing technique for full-text database systems. In: Proceedings of the 18th International Conference on Very Large Databases. (1992) 352–362
4. Tomasic, A., Garcia-Molina, H., Shoens, K.: Incremental updates of inverted lists for text document retrieval. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. (1994) 289–300
5. Baeza-Yates, R., Ribeiro-Neto, B.A.: Modern information retrieval. Addison-Wesley Publishing (1999)
6. Tomasic, A., Garcia-Molina, H.: Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In: Proceedings of the International Conference on Parallel and Distributed Information Systems. (1992) 8–17
7. Jeong, B.S., Omiecinski, E.: Inverted file partitioning schemes in multiple disk systems. IEEE Transactions on Parallel and Distributed Systems **6**(2) (1995) 142–153
8. Ribeiro-Neto, B.A., Barbosa, R.A.: Query performance for tightly coupled distributed digital libraries. In: Proceedings of the Third ACM Conference on Digital Libraries. (1998) 182–190
9. Cambazoglu, B.B., Aykanat, C.: Performance of query processing implementations in ranking-based text retrieval systems using inverted indices. Information Processing and Management **42**(4) (2005) 875–898
10. Burns, G., Daoud, R., Vaigl, J.: LAM: An Open Cluster Environment for MPI. In: Proceedings of the Supercomputing Symposium. (1994) 379–386