Memory Resident Parallel Inverted Index Construction

Tayfun Kucukyilmaz, Ata Turk and Cevdet Aykanat

Abstract Advances in cloud computing, 64-bit architectures and huge RAMs enable performing many search related tasks in memory.We argue that term-based partitioned parallel inverted index construction is among such tasks, and provide an efficient parallel framework that achieves this task. We show that by utilizing an efficient bucketing scheme we can eliminate the need for the generation of a global index and reduce the communication overhead without disturbing balancing constraint. We also propose and investigate assignment schemes that can further reduce communication overheads without disturbing balancing constraints. The conducted experiments indicate promising results.

1 Introduction

Inverted index is the de-facto data structure used in state-of-the-art text retrieval systems. Even though it is quite simple as a data structure, Web-scale generation of a global inverted index is very costly [2]. Since the data to be indexed is crawled and stored by distributed or parallel systems (due to performance and scalability reasons), parallel index construction techniques are essential. Despite the popularity of document-based partitioned inverted indices, term-based partitioning has advantages that can be exploited for better query processing [4].

The following studies on index construction [3, 5, 6] extend disk-based techniques for parallel systems. In [5, 6], authors propose a parallel disk-based algorithm with a centralized approach to generate the global vocabulary. In [5] authors analyze the merging phase of the inverted lists and present three algorithms. In [3], authors start from a document partitioned collection and proposed a software-pipelined inversion architecture.

T. Kucukyilmaz · A. Turk · C. Aykanat (✉)
Computer Engineering Department, Bilkent University, 06800 Ankara, Turkey
e-mail: aykanat@cs.bilkent.edu.tr

E. Gelenbe et al. (eds.), *Computer and Information Sciences II*,
DOI: 10.1007/978-1-4471-2155-8_12,

With the advent of 64-bit architectures, huge memory spaces are available to single machines and even very large inverted indices can fit into the total distributed memory of a cluster of such systems, enabling memory-based index construction. Given the current advances in network technologies and cloud computing and the high availability of low cost memory, in-memory solutions for parallel index construction should be considered seriously.

In this work, we propose an efficient, memory-based, parallel index construction framework for generating term-based partitioned inverted indices starting from a document-based partitioned collection (possibly due to parallel crawling). In this framework, we propose to mask the communication costs associated with global vocabulary construction and communication with a term-to-bucket assignment schema. Furthermore, we investigate several assignment heuristics for improving both the final storage balance and the communication costs of inverted index construction. Here, storage balance is important since it relates to query processing loads of processors, whereas the communication cost is important since it determines the running time of parallel inversion. Our contributions in this work are prior to optimizations such as compression [7].

2 Parallel Inversion

Our overall parallel inversion scheme has the following phases: local inverted index construction, term-to-processor assignment, and inverted list exchange and merge. In this section we describe these three phases in detail.

The first task in our framework is generating local inverted indices for all local document collections on each processor. As each processor contains a non-overlapping portion of the whole document collection, this operation can be achieved concurrently without communication.

After local inverted index construction phase, a term-to-processor assignment phase has to follow. In order to achieve a term-to-processor assignment, normally a global vocabulary has to be generated. This could be done by sending each term string, in its word form, to a host processor, where a global vocabulary is constructed. However in such a scheme, a particular term would be sent to the host machine by all processors if all processors contain that specific term. Thus we propose to group terms into a fixed number of buckets prior to the term-to-processor assignment. Using hashing, each word in a local vocabulary is assigned to a bucket. Afterwards, a host processor computes a bucket-to-processor assignment and broadcasts this information to the processors.

At the end of local inversion phase, each processor has a local vocabulary and a set of inverted lists for its terms. However, different processors may contain different portions of an inverted list for each term. For the final term-based partitioned inverted index to be created, the inverted list portions of each term should be accumulated to a single processor. To this end, each term in the global vocabulary should be assigned to a particular processor. This term-to-processor

assignment depicts an inverted index partitioning problem. Many different criteria can be considered when finding a suitable index partitioning, but we focus on balancing the storage loads of processors and minimizing the communication overhead of the inversion process. The storage balance guarantees an evenly distribution of the final inverted index. As the memory is assumed to be limited throughout this work, with an even distribution of the storage loads, larger indices can fit in the same set of processors. Storage balancing is also expected to infer balancing on the query processing loads of the processors. Since inversion is a highly communication-bound process, the minimization of the communication overhead ensures that the inverted list exchange phase of the parallel inversion process takes less time. In this work, we model the minimization of the communication overhead as the minimization of the total communication volume while maintaining balance on communication loads of processors.

At the end of the bucket-to-processor assignment phase, all assignments are broadcast to processors and processors exchange their partial inverted lists in an all-to-all fashion. When sending the local inverted lists to their assigned processor, the vocabulary should also be sent since processors do not necessarily contain all vocabulary terms and do not know which terms will be retrieved from other processors. Although such a communication incur additional costs, since the processor-to-host bottleneck due to global vocabulary construction is already avoided, this additional communication cost is easily compensated.

The inverted-list exchange between processors is achieved in two steps. In the first step, the terms (in word form) and their posting sizes are communicated. At the end of this step, all processors obtain their final local vocabularies and can reserve space for their final local inverted index structures. Then the inverted lists are exchanged. At the end of inverted list exchange, posting lists for each term are merged and written into their reserved spaces in local inverted indices.

3 Bucket-to-Processor Assignment Schemes

In the forthcoming discussions we use the following notations: The vocabulary of terms is indicated with $\mathcal{T}$. The posting list of each term $t_j \in \mathcal{T}$ is distributed among the K processors. $w_k(t_j)$ denotes the size of the posting list portion of term t_j that resides in processor p_k at the beginning of the inversion, whereas $w_{tot}(t_j) = \sum_{k=1}^{K} w_k(t_j)$ denotes the total posting list size of term t_j.

We assume that prior to bucket-to-processor assignment, each processor has built its local inverted index $\mathcal{I}_k$ and partitioned the vocabulary $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ containing n terms, into a predetermined number m of buckets. The number of buckets m is selected such that $m \ll n$ and $m \gg K$. Let $\mathcal{B} = \Pi(\mathcal{T}) = \{\mathcal{T}_1 = b_1, \mathcal{T}_2 = b_2, \ldots, \mathcal{T}_m = b_m\}$ denote a random term-to-bucket partition, where $\mathcal{T}_i$ denotes the set of terms that are assigned to bucket b_i. In $\mathcal{B}$, $w_{tot}(b_i)$ denotes the total size of the posting lists of terms that belong to b_i.

In an m-bucket and K-processor system, the bucket-to-processor assignment can be represented via a K-way partition $\Pi(\mathcal{B}) = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k\}$ of the buckets among the processors. In $\Pi(\mathcal{B})$, $w_k(b_i)$ denotes the total size of the posting list portions of terms that belong to b_i and reside in processor p_k at the beginning of the inversion. The performance of a bucket-to-processor assignment is measured in terms of two metrics. The storage load balance and the communication cost. The storage load $S(p_k)$ of a processor p_k induced by the assignment $\Pi(B)$ is defined as follows:

$$S(p_k) = \sum_{b_i \in \mathcal{B}_k} \sum_{t_j \in b_i} w_{tot}(t_j). \tag{1}$$

The communication cost of a processor p_k induced by assignment $\Pi(B)$ has two components. Each processor must receive all portions of the buckets assigned to itself from other processors. Thus total receive cost/volume of a processor p_k is:

$$Recv(p_k) = \sum_{b_i \in \mathcal{B}_k} \sum_{t_j \in b_i} (w_{tot}(t_j) - w_k(t_j)). \tag{2}$$

Each processor also sends all postings that are not assigned to it to some other processor. The total send cost of p_k is:

$$Send(p_k) = \sum_{b_i \notin \mathcal{B}_k} \sum_{t_j \in b_i} w_k(t_j) \tag{3}$$

Total communication cost of a processor is defined as the sum of its send and receive costs.

The MCA scheme is based on the following simple observation [1]. If we assign each bucket $b_i \in \mathcal{B}$ to processor p_k that has the largest $w_k(b_i)$ value, we will achieve an assignment with globally minimum total communication volume.

The BLMCA scheme incorporates a storage balance heuristic to MCA [1]. This scheme works in an iterative manner assigning one bucket to a processor at a time. For each bucket, first the processor that will cause the minimum total communication is determined using MCA scheme. If this processor is not the bottleneck processor (in terms of storage load) at that iteration, the bucket is assigned to that processor. Otherwise, the bucket is assigned to the minimally loaded processor.

In BLMCA, two cost metrics, storage load balance and communication cost are calculated and at each iteration an assignment decision that optimizes only one of these metrics is made. The decisions of MCA and BLMCA regarding the communication cost minimization only optimizes the total communication cost and ignores the maximum communication cost of a processor. In order to minimize maximum communication cost, we should consider both the receive cost of the assigned processor and the send costs of all other processors.

To this end we define the energy E of an assignment $\Pi(B)$ based on the storage loads and communication costs of processors. We define two different energy functions for a given term-to-processor assignment $\Pi(B)$:

$$E^1(\Pi(B)) = Max\{Max_{1 \le k \le K}\{Comm(p_k)\}, Max_{1 \le k \le K}\{S(p_k)\}\} \quad (4)$$

$$E^2(\Pi(B)) = \sum_{1}^{K}(Comm(p_k))^2 + \sum_{1}^{K}(S(p_k))^2 \quad (5)$$

Utilizing these energy functions, we propose a constructive algorithm that assigns buckets to processors one-by-one. The energy increase in the system by K possible assignments of each bucket are considered, and the assignment that incurs the minimum energy increase is performed. That is, for the assignment of a bucket b_i in the given order, we select the assignment that minimizes $E(\Pi(B_{i-1} \cup \{b_i\})) - E(\Pi(B_{i-1}))$, where B_{i-1} denotes the set of already assigned buckets. We call E^1-based and E^2-based assignment schemes as E^1A and E^2A respectively in our experiments.

4 Experiments

In order to test the performance of the proposed assignment schemes for parallel inversion, we conducted two types of experiments. The first set of experiments are simulations to report on the storage imbalance and communication volume performances of the assignment schemes. The second set of experiments are actual parallel inversion runs provided in order to show how improvements in performance metrics relate to parallel running times. These experiments are conducted on a PC-cluster with $K = 32$ nodes, where each node is an Intel Pentium IV 3.0 GHz processors with 1 GB RAM connected via an interconnection network of 100 Mb/sec fast ethernet.

We conducted our experiments on a realistic dataset obtained by crawling educational sites across America. The raw size of the dataset is 30 GB and contains 1,883,037 pages from 18,997 different sites. The biggest site contains 10,352 pages while average number of pages per site is 99.1. The vocabulary of the dataset consists of 3,325,075 distinct terms. There are 787,221,668 words in the dataset. The size of the inverted index generated from the dataset is 2.8 GB.

Tables 1 and 2 compare the storage load balancing and communication performances of the assignment schemes for $K = \{4, 8, 16, 32, 64, 128\}$. We also implemented a random assignment (RA) algorithm, which assigns buckets to processors randomly, as a baseline assignment scheme. As seen in Table 1, MCA achieves the worst final storage imbalance. This is expected since MCA considers only minimization of the total communication cost, disregarding storage balance and as seen in Table 2 MCA achieves lowest average communication cost. BLMCA algorithm on the other hand, achieves best final storage imbalance. This is also expected since the primary objective of BLMCA is to balance the processor loads during the assignments instead of minimizing the communication costs. As seen in Table 2, this storage balancing performance is achieved at the expense of higher average communication values per processor. Experiments indicate that

Table 1 Percent load imbalance values

	Initial	Final				
K		RA	MCA	BLMCA	E^1A	E^2A
4	4.4	12.1	38.3	0.0	6.1	5.5
8	11.7	09.9	60.0	0.1	18.2	14.4
16	18.2	27.4	66.2	1.7	27.2	20.0
32	44.1	29.6	83.0	5.4	35.2	31.1
40	32.2	37.0	77.4	6.2	38.4	31.4
64	44.7	56.6	92.2	11.5	46.9	33.7
128	65.3	94.7	95.6	15.7	64.1	40.4

Table 2 Message volume (send + receive) per processor (in terms of $\times 10^6$ postings)

	RA		MCA		BLMCA		E^1A		E^2A	
K	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
4	131.19	145.71	122.09	150.26	127.45	128.62	124.76	125.29	131.24	131.36
8	76.55	90.58	71.45	119.75	73.40	75.97	72.02	74.53	76.67	76.79
16	41.01	49.25	38.32	77.11	39.22	43.44	38.52	42.33	41.60	41.66
32	21.19	28.75	19.82	71.13	20.28	26.03	19.94	25.08	20.54	21.61
40	17.05	23.96	15.99	44.79	16.32	20.01	16.03	19.22	17.04	17.71
64	10.76	17.77	10.09	74.27	10.34	15.42	10.15	14.75	10.86	11.89
128	5.42	11.97	5.09	65.59	5.22	10.98	5.12	10.02	6.81	7.95

Table 3 Parallel inversion times (in seconds) including assignment and inverted list exchange times for different assignment schemes

K	RA	MCA	BLMCA	E^1A	E^2A
4	69.19	81.34	68.63	68.67	68.49
8	51.42	66.76	46.45	46.59	45.74
16	35.89	60.82	33.04	32.90	32.48
32	19.31	49.45	18.20	17.91	17.20

E^1A and E^2A algorithms both achieve reasonable storage load balance that are either close or better than the performance of RA scheme. Also as seen in Table 2, for K values higher than 8, E^2A achieves the lowest maximum communication volumes. Table 2 also indicates that the average and maximum communication costs induced by E^2A are very close, which means that E^2A manages to distribute the communication loads among processors evenly.

Table 3 shows the running times of our parallel memory-based index inversion algorithm under different assignment schemes. In this table, it is assumed that the local inverted indices are already created and the time for this operation is neglected. As expected from the results presented in Table 1 and Table 2, MCA induces the highest inversion time, RA, BLMCA, E^1A, and E^2A induce similar inversion times and the E^2A scheme achieves the lowest inversion times.

5 Conclusions

In this paper, a memory-based parallel inverted index construction framework was examined. An extensive step-by-step experimentation of our model was presented and further insight were provided using theoretical results and simulations. Also, several problems involving the creation of this framework were identified.

References

1. Aykanat, C., Cambazoglu, B.B., Findik, F., Kurc, T.: Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. J. Parallel Distrib. Comput. **67**, 77–99 (2007)
2. Cho, J., Garcia-Molina, H.: The evolution of the web and implications for an incremental crawler. In: Proceedings of the 26th International Conference on VLDB (2000)
3. Melink, S., Raghavan, S., Yang, B., Garcia-Molina, H.: Building a distributed full-text index for the web. ACM Trans. Inf. Syst. **19**, 217–241 (2001)
4. Moffat, A., Webber, W., Zobel, J.: Load balancing for term-distributed parallel retrieval. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 348–355 (2006)
5. Ribeiro-Neto, B., Moura, E.S., Neubert, M.S., Ziviani, N.: Efficient distributed algorithms to build inverted files. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in IR, pp. 105–112 (1999)
6. Ribeiro-Neto, B.A., Kitajima, J.P., Navarro, G., Sant'Ana, C.R.G., Ziviani, N.: Parallel generation of inverted files for distributed text collections. In: Proceedings of the 18th International Conference of the Chilean Computer Science Society (1998)
7. Zobel, J., Moffat, A., Ramamohanarao, K.: Inverted files versus signature files for text indexing. ACM Trans. Database Syst. **23**, 453–490 (1998)