Scheduling Intersection Queries in Term Partitioned Inverted Files

Mauricio Marin[1], Carlos Gomez-Pantoja[2],
Senen Gonzalez[2], and Veronica Gil-Costa[3]

[1] Yahoo! Research, Santiago, Chile
[2] University of Chile
[3] University of San Luis, Argentina

Abstract. This paper proposes and presents a comparison of scheduling algorithms applied to the context of load balancing the query traffic on distributed inverted files. We put emphasis on queries requiring intersection of posting lists, which is a very demanding case for the term partitioned inverted file and a case in which the document partitioned inverted file used by current search engines can perform very efficiently. We show that with proper scheduling of queries the term partitioned approach can outperform the document partitioned approach.

1 Introduction

Cluster based search engines use distributed inverted files [13] for dealing efficiently with high traffic of user queries. An inverted file is composed of a vocabulary table and a set of posting lists. The vocabulary table contains the set of relevant terms found in the text collection. Each of these terms is associated with a posting list which contains the document identifiers where the term appears in the collection along with additional data used for ranking purposes. To solve a query, it is necessary to get the set of documents associated with the query terms and then perform a ranking of these documents in order to select the top R documents as the query answer.

The approach used by well-known Web search engines to the parallelization of inverted files is pragmatic, namely they use the document partitioned approach. Documents are evenly distributed on P processors and an independent inverted file is constructed for each of the P sets of documents. The disadvantage is that each user query has to be sent to the P processors and it can present imbalance at posting lists level (this increases disk access and interprocessor communication costs). The advantage is that document partitioned indexes are easy to maintain since insertion of new documents can be done locally and this locality is extremely convenient for the posting list intersection operations required to solve the queries (they come for free in terms of communication costs). Intersection of posting lists is necessary to determine the set of documents that contain all of the terms present in a given user query.

Another competing approach is the term partitioned index in which a single inverted file is constructed from the whole text collection to then distribute

E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 434–443, 2008.
© Springer-Verlag Berlin Heidelberg 2008

evenly the terms with their respective posting lists onto the processors. However, the term partitioned inverted file destroys the possibility of computing intersections for free in terms of communication cost and thereby one is compelled to use strategies such as smart distribution of terms onto processors to increase locality for most frequent terms (which can be detrimental to overall load balance) and caching. However, it is not necessary to broadcast queries to all processors (which reduces communication costs) and latency disk costs are smaller as they are paid once per posting list retrieval per query, and it is well-known that in current cluster technology it is faster to transfer blocks of bytes through the interprocessors network than from Ram to Disk. Nevertheless, the load balance is sensitive to queries referring to particular terms with high frequency, making it necessary to use posting lists caching strategies to overcome imbalance in disk accesses.

Both strategies are efficient depending on the method used to perform the final ranking of documents. In particular, the term partitioned index is better suited for methods that do not require performing posting list intersections. For this case we have observed that the balance of disk accesses, that is posting list fetching, and document ranking are the most relevant factors affecting the performance of query processing. The balance of interprocessors communication depends on the balance of these two components. From empirical evidence we have observed that *moderate* imbalance in communication is not detrimental to performance.

The scenario for the term partitioned index is completely different for queries requiring the intersection of posting lists. This can become extremely expensive in communication because it is necessary to send posting lists between processors to let them being intersected. From previous work on implementing search engines on P2P networks we can learn of a number of attempts to reduce the size of the posting lists sent to other processors to be intersected [3, 8, 11]. In addition, work on smart distribution of terms devised to increase the probability of terms appearing very frequently in queries being located in the same processor have been presented in [1, 2, 6, 7, 12]. Other approaches avoid the imbalance effect of frequent terms by just replicating their posting lists in two or more processors [6, 12].

The most recent study on the comparative performance of the term and document partitioned index is the work presented in [6]. They use a realization of the term partitioned index called the *pipelined* approach. In this scheme a query traverses one by one the processors that contain query terms and in each visit it tries to determine the best ranked document that can become part of the top ranked ones to be presented to the user. Since the partial ranking of document is tied to the processors that contain query terms, the imbalance can become significant because of the terms appearing in queries very frequently. Their solution is a combination of posting list replication and the least loaded processor first heuristic for improving load balance. In the experiments they use queries that do not require intersection of posting lists (OR queries). Their study concludes that the term partitioned index cannot outperform the document partitioned index.

However, in [4] we show that for this type of queries the term partitioned index can outperform significantly to the document partitioned index. This because we de-couple document ranking from the processors where the query terms are located and our method of ranking does not require large portions of posting lists to calculate the top R documents. We also show under our method that the most simple heuristics for load balancing leads to running times as good as those achieved by more sophisticated strategies for static and dynamic scheduling of tasks onto processors.

Now the challenge is to consider the more complicated case of queries requiring posting list intersections (AND queries). In this case the amount of communication can be very large and to avoid this one is compelled to consider performing intersection and ranking in the processors holding query terms. Certainly communication can be reduced by using the pruning techniques mentioned in [9], but it is a matter of how big is the text collection to be in the same situation of high cost in communication.

In this paper we deal with this problem and propose solutions based on query scheduling that are able to outperform the document partitioned index under situations of high traffic of queries. A contribution of this paper is also the way in which we model the parallel processing of queries to support scheduling decisions. Queries arrive to the processors from a receptionist machine that we call the *broker*. We study the case in which the broker is responsible for assigning the work to the processors. Jobs badly scheduled onto the processors can result in high imbalance. To this end the broker uses a scheduling algorithm. For example, a simple approach is to distribute the queries uniformly at random onto the processors in a blind manner, namely just as they arrive to the broker they are scheduled in a circular round-robin manner. A more sophisticated scheduling strategy demands more computing power from the broker so in our view this cost should be paid only if load balance improves significantly.

2 Scheduling Framework

The broker simulates the operation of a search engine that is processing the queries as follows. First, it uses bulk-synchronous parallel (BSP) computing to process queries. In BSP [10] the computation is organized as a sequence of *supersteps*. During a superstep, the processors may perform computations on local data and/or send messages to other processors. The messages are available for processing at their destinations by the next superstep, and each superstep is ended with the barrier synchronization of the processors. The underlying communication library ensures that all messages are available at their destinations before starting the next superstep.

Secondly, query processing is divided in “atoms” of size K, where $K = 2R$ where R is the number of documents presented to the user as part of the query answer. These atoms are scheduled in a round-robin manner across supersteps and processors. The asynchronous tasks are given K sized quanta of processor

time, communication network and disk accesses. These quanta are granted during supersteps, namely they are processed in a bulk-synchronous manner.

As all atoms are equally sized then the net effect is that no particular task can restrain others from using the resources. This because (**i**) computing the solution to a given query can take the processing of several atoms, (**ii**) the search engine can start the processing of a new query as soon as any query is finished, and (**iii**) the processors are barrier synchronized and all messages are delivered in their destinations at the end of each superstep. It is not difficult to see that this scheme is optimal provided that we find an "atom" packing strategy that produces optimal load balance and minimizes the total number of supersteps required to complete a given set of queries (this is directly related to the critical path for the set of queries). Relevant literature on BSP has shown that BSP computations can simulate well asynchronous computations such as those performed by current search engines. In [5] we show that the difference in performance between these two modes of computation is related to the query traffic.

The simulation assumes that at the beginning of each superstep the processors get into their input message queues both new queries placed there by the broker and messages with pieces of posting lists related to the processing of queries which arrived at previous supersteps. The processing of a given query can take two or more supersteps to be completed. The processor in which a given query arrives is called the *ranker* for that query since it is in this processor where the associated document ranking is performed.

Every query is processed using two major steps: the first one consists on fetching a K-sized piece of every posting list involved in the query and sending them to the ranker processor. In the second step, the ranker performs the actual ranking of documents and, if necessary, it asks for additional K-sized pieces of the posting lists in order to produce the K best ranked documents that are passed to the broker as the query results. We call this *iterations*. Thus the ranking process can take one or more iterations to finish. In every iteration a new piece of K pairs (doc_id, frequency) from posting lists are sent to the ranker for every term involved in the query. At a given interval of time, the ranking of two or more queries can take place in parallel at different processors along with the fetching of K-sized pieces of posting lists associated with new queries. For AND queries it is necessary to first make the intersection of all involved posting lists to determine the document containing all query terms. Then the ranking can proceed as described above by considering only the documents in the intersection.

In figure 1 we illustrate the scheduling problem under this bulk-synchronous framework. The broker performs the scheduling maintaining two windows which account for the processors work-load through the supersteps. One window is for the intersection + ranking operations effected per processor per superstep and the another is for the posting list fetches from disk also per processor per superstep. In each window cell we keep the count of the number of operations of each type. The optimization goal is to achieve an imbalance of about 15%

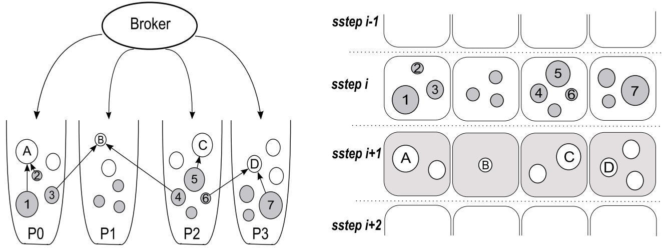

Fig. 1. Scheduling problem in the term partitioned inverted file. Grey balls represent inverted lists of different length and white balls represent the cost of intersecting the posting lists. The abstraction in supersteps allows the broker to consider the temporality of operations and model the cost of the tradeoff involved in decisions such as where to schedule the intersection and ranking, and at index construction time where to place terms onto processors.

across processors. Imbalance is measured via efficiency, which for a measure X is defined by the ratio average(X)/maximum$(X) \leq 1$, over the P processors.

3 Description and Evaluation of Scheduling Algorithms

In the experimental results presented in this section we have used a 12GB sample of the Chilean Web and query log from `www.todocl.cl`. We use this text collection to explore the effect of load imbalance as we keep fixed the overall size of the inverted file and we increase the number of processors P. We performed experiments by running BSPonMPI programs on a cluster with 32 processors. In every run we process 10,000 queries in *each* processor. That is the total number of queries processed in each experiment reported below is $10{,}000 \times P$. Thus running times are expected to grow with P since the communication hardware has at least $\log P$ scalability. Another parameter used in our experiments is the query traffic Q which indicates the rate of queries per unit time in each processor and superstep.

Thus in the figures shown below the curves for 32 processors are higher in running time than the ones for 4 processors. We have found this useful to see the efficiency of the different strategies in the sense of how well they support the inclusion of more processors to work on a problem of a fixed size N. Given the size of text collection, the case for $P = 32$ becomes a very demanding one in terms of the small slackness N/P available to load balance processors. We show running measures normalized between 0 and 1 by dividing all running times by the maximum running time among the set of strategies being compared in the figure. This makes it easier to see the relative difference in performance of the different strategies. Overall the average running time per query of our programs is less than 1ms for the 12GB sample. Below we refer to the term partitioned index as Global and the document partitioned index as Local.

We have implemented all strategies reported in the literature as the most efficient ones in aspects related to smart term distribution onto processors and

replication. We report running times obtained with the ones that behaved best. We also contribute with scheduling algorithms based on the bulk-synchronous abstraction of the scheduling problem described in the previous section. The percentage of replication is 20% which is very similar to the analysis reported in [6]. The strategies tested are the following.

Round-Robin [Circular]: terms are distributed onto machines such that the n-th term goes to machine $n\%P$. Duplication is performed assigning a processor selected at random for the most frequent terms to be replicated.

Correlation-Aware [Corr]: the frequency of each pair of terms is retrieved from the query logs which acts as a training set. Then, all pair-terms are ordered decreasingly according to its frequency. The idea is that two terms highly correlated should be assigned to the same machine. If one of the terms was already distributed, then the other term is assigned in the same machine. If the entire pair has not been distributed, then the pair is assigned to the least loaded machine. We have used two variations: either P or P^2 most frequent terms are distributed –one per machine– before distributing the aforementioned pairs. Note that the initial distribution given as input, only is used to obtain the load of each machine and the processors where is not a certain term. This strategy is similar to the mentioned in [1, 2].

Fill-Smallest: the technique introduced in [6] sorts the terms decreasingly according to the load L_t, defined to be the length of the inverted list of term t times the frequency of t in the query log. In practice, this function produces a big imbalance we considered two variations, one in which the weight of the term is the length of the list, and the other in which the weight is the frequency. The same technique is used for replicating terms.

List-Driven and Node-Driven: these strategies have been developed to properly replicate inverted lists [12]. List-Driven selects first a term to replicate, and then a machine to maximize the benefit associated with this replication. This benefit involves the communication costs associated with the queries of the log. Analogously, the Node-driven approach select in each iteration the machine less loaded, and then finds the term whose allocation to that machine that maximizes the benefit. The input consists of a distribution of terms without replication. For this the authors propose two methods. The first one using a random distribution, and the second guided by a graph partitioning software. Our experiments used the first option. According to them, List-Driven seems to perform better.

Figure 2 shows the normalized running time for the different strategies with $Q = 32$. Note that for the Correlation-aware and Fill-Smallest algorithms there are many variations in performance for different tuning of their operational setting. We have chosen only the versions of each algorithm with the best performance, and these are the ones shown in the figure.

In the figure 3 we show results for the case of posting lists replication. The figure shows poor improvements in running time, thus we do not consider these techniques any further in this paper.

In the term-partitioning approach we can schedule intersections from the broker to improve balance. The plain strategy, called Global, takes into account

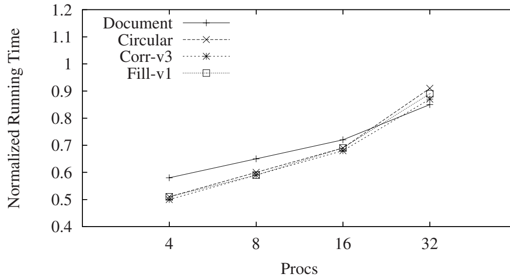

Fig. 2. Comparison of term distribution strategies without replication with Q=32

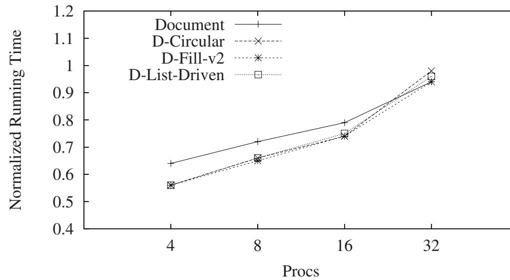

Fig. 3. Comparison of term distribution strategies with replication with Q=32

the communication cost so that to solve a query of two or more terms the broker chooses the machine where the largest term (size of its inverted list) was assigned. In the case of queries with one term, only the ranking operation is assigned to the machine that received the query. We implemented three additional strategies to evaluate different alternatives for this basic scheme.

In the first one (Scheduling-1), we schedule the intersections in a Round-Robin manner, such that in each superstep each machine has the same number of intersection and ranking operations (each ranking operation is performed in the same machine of the intersection to avoid additional communication costs). In the second strategy (Scheduling-2) we select the machines that have the terms involved in the query, then we calculate their load and the query is assigned to the least loaded processor. The load of a machine is determined from the sizes of the terms involved in the solution of the queries assigned to that machine. In the last strategy (Scheduling-3) we give more degrees of freedom to the second strategy, i.e., we mean that the possible candidates to assign a query are all the processors, not only the processors involved in the query. Then, we use a simple heuristic that chooses some queries to be moved to other machines in function of their load. The results are shown in the figure 4.

The previous figure shows clearly that a little effort in assigning intelligently the intersections can be important to reducing overall running time. Moreover when there are a large degree of imbalance we observed that the difference was approximately 30%. The source of this differences is the communication cost.

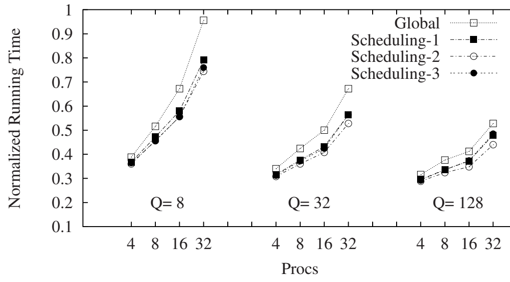

Fig. 4. Scheduling of intersection operations

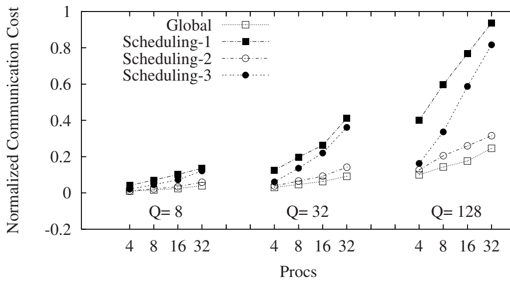

Fig. 5. Communication cost of query scheduling strategies

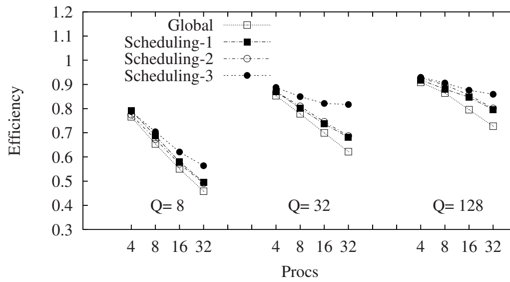

Fig. 6. Efficiency at superstep running time level

The strategy Scheduling-2 could be viewed as an improvement of the Global strategy. The machines involved in the query solution are the same with the exception that Scheduling-2 distributes the intersections to the machines that have minimum load.

In the figure 5 we show the results for the overall cost of communication. Global strategy can be seen as the optimal strategy in terms of communication costs because it only communicates the shortest list related to each query. The Scheduling-2 strategy is very similar in terms of communication.

On the other hand, figure 6 shows the efficiency at computing time level. We measured in each superstep the running time of each machine, but without

considering communication overhead. Then we obtained the efficiency measured as the maximum running time at each superstep divided by the average running time. The figure shows that the most balanced strategy is Global with Scheduling 3. However this is not enough to achieve good performance as the effect of communication is relevant for intersection operations.

3.1 Document Versus Best-Term Partitioned Strategies

For the term partitioned approach, we used two schemes for distributing the terms. For the Global strategy explained above, we use the Correlation-Aware distribution in its P^2 version (G-Corr-v3). In addition for the Scheduling-2 (the best of all three) we use the Fill-Smallest partitioning in its *size* version (S-Fill-v1). We also compare with the document partitioning index (Local).

We executed under different traffic conditions (Q) and number of processors. In figure 7 we show the results. There we can observe that when there is low query traffic, the Local strategy outperforms the Global ones. But when we have high traffic of queries, the Global strategies outperform the Local one.

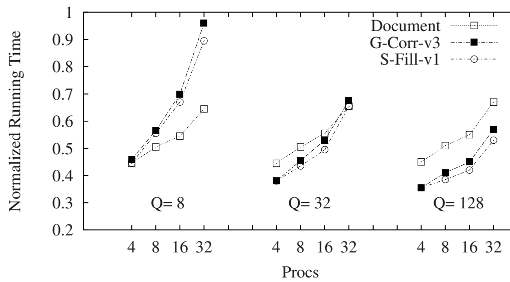

Fig. 7. Normalized running time

4 Conclusions

We have evaluated scheduling strategies which show that the term partitioned inverted file can outperform the document partitioned one under the requirement of posting list intersections. However, this can only occur in situations of high traffic of queries case in which overall load balance is good enough. Nevertheless, for low traffic the cluster resources are not being fully utilized so we think this is not a serious drawback since response times are still small despite of imbalance and high communication.

Notice that we did not resort to techniques for reducing communication such as those based on Bloom filters. Neither we considered compression of posting lists so the results of this paper are pessimistic for the term partitioned index since there is still room for further optimizations. In addition, we have only considered sending the whole posting list in order to perform exhaustive intersection. However, in large scale systems it is certainly possible to avoid this by

employing pruning strategies. The rationale is that for each query term we only need to get the top-K documents in the intersection and not the whole set of documents in the intersection set.

Acknowledgment. Partially funded by Millennium Nucleus CWR, Grant P04-067-F, Mideplan, Chile.

References

1. Chaudhuri, S., Church, K., König, A.C., Sui, L.: Heavy-tailed distributions and multi-keyword queries. In: SIGIR 2007: 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 663–670. ACM, New York (2007)
2. Falchi, F., Gennaro, C., Rabitti, F., Zezula, P.: Mining query logs to optimize index partitioning in parallel web search engines. In: INFOSCALE 2007: 2nd International Conference on Scalable Information Systems (2007)
3. Li, J., Loo, B., Hellerstein, J., Kaashoek, F., Karger, D., Morris, R.: The feasibility of peer-to-peer web indexing and search (2003)
4. Marin, M., Gomez, C.: Load balancing distributed inverted files. In: WIDM 2007: 9th annual ACM international workshop on Web information and data management, pp. 57–64. ACM, New York (2007)
5. Marin, M., Costa, V.G. (SyncjAsync)$^+$ MPI search engines. In: Cappello, F., Hrault, T., Dongarra, J. (eds.) PVM/MPI 2007. LNCS, vol. 4757, pp. 117–124. Springer, Heidelberg (2007)
6. Moffat, A., Webber, W., Zobel, J.: Load balancing for term-distributed parallel retrieval. In: SIGIR 2006: 29th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 348–355. ACM, New York (2006)
7. Moffat, W., Zobel, J.W., Baeza-Yates, R.: A pipelined architecture for distributed text query evaluation. Information Retrieval (August 2007)
8. Reynolds, P., Vahdat, A.: Efficient peer-to-peer keyword searching. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 21–40. Springer, Heidelberg (2003)
9. Suel, T., Mathur, C., wen Wu, J., Zhang, J., Delis, A., Kharrazi, M., Long, X., Shanmugasundaram, K.: ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In: WWW 2003: 12th International World Wide Web Conference (2003)
10. Valiant, L.: A bridging model for parallel computation. Comm. ACM 33, 103–111 (1990)
11. Zhang, J., Long, X., Suel, T.: Performance of compressed inverted list caching in search engines. In: WWW 2008: 17th International World Wide Web Conference (2008)
12. Zhang, J., Suel, T.: Optimized inverted list assignment in distributed search engine architectures. In: IPDPS 2007: 23rd IEEE International Parallel and Distributed Processing Symposium (2007)
13. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys 38(2) (2006)