Load Balancing Distributed Inverted Files: Query Ranking

Carlos Gomez-Pantoja Mauricio Marin
Yahoo! Research, Santiago
University of Chile

Abstract

Search engines use inverted files as index data structures to speed up the solution of user queries. The index is distributed on a set of processors forming a cluster of computers and queries are received by a broker machine and scheduled for solution in the cluster. The broker must use a scheduling algorithm to assign queries to processors since the computations associated with the ranking of documents that form part of the solutions to queries can take a significant fraction of the total running time. The cost of this task can be highly variable and depends on the particular user preferences for words when formulating queries in a given period of time. Thus the scheduling algorithm must be able to cope efficiently with a highly dynamic and very large amount of jobs being assigned in an on-line manner to the processors. In this paper we evaluate a number of scheduling algorithms proposed in the literature in the context of scheduling queries on a search engine.

1 Introduction

Search engines account for about 5% of the time spent by hundreds of millions of users on the Web. They are systems devised to support thousands of queries per second which are executed on billions of Web documents. Query response times are kept small by properly indexing data and distributing them on thousands of Linux boxes forming clusters of computers. To this end, the algorithmic design and implementation of current Web Search Engines is based on the asynchronous message passing approach to parallel computing in which each newly arriving query is serviced by an independent thread in a classical multiple masters/slaves scheme.

On the other hand, the amount of work demanded by the solution of queries follows the so-called Zipf's law which in practice means that some queries, in particular the ones composed of most popular terms, can demand large amounts of processing times whereas others containing less frequent terms can require a comparatively much smaller processing time. Thus under this asynchronous approach and hardware latencies a given query can easily restrain smaller queries by consuming comparatively larger amounts of resources in processor cycles, disk and interprocessors network bandwidths.

This makes a case for proper consideration of the way in which the operations related to the solution of queries are scheduled onto the cluster processors. We have found that a careful design of the major steps involved in the processing of queries can allow its decomposition in such a way that we can let every query share the cluster resources evenly in a round-robin manner. We have observed that this scheme can be particularly useful in preventing unstable behavior under unpredictable variations in the query traffic arriving to the search engine. We implement this round-robin scheme on top of the bulk-synchronous model of parallel computing [19] though a semi-synchronous MPI approach is also feasible with similar performance.

In particular, we have observed for the standard asynchronous method of query processing that sudden peaks in the query traffic can be very detrimental to overall performance due to the Zipf's law distribution of the workload per query. We have also observed that the round-robin method of query processing solves this problem efficiently and the reason comes from the fact that each query is granted an equally sized fraction of the resources. However, in the Sync mode and for low traffic of queries it is not efficient to barrier synchronize processors and pack together messages for just a few queries being processed into the cluster as required in the bulk-synchronous model [19].

Thus in [17] we proposed a method to allow the search engine to automatically switch between the Async and Sync modes of operation. In the Async mode each query is serviced by a different thread which in turn send messages to threads located in other processors in order to gather the required information to produce the respective answer. In the Sync mode the number of threads is reduced drastically and query processing is performed in batches. In this case we decompose the solution of queries in iterations and assign to each query a similar amount of resources in each iteration.

In [18] we proposed a method to schedule query process-

ing under a search engine operating in the Sync mode. In this paper we extend those results for the case of a search engines operating in the Async mode. A reasonable conjecture is that the bulk-synchronous method presented in [18] to predict running time and schedule queries can also be applied to the Async mode. However we have not studied it in detail so far. Here the underlying assumption is that the model can also predict fairly well the average work performed by the Async machine. Nevertheless, as queries in the Async mode can be seen as independent jobs, namely they do not interfere each other in terms of resource competition in a significant manner under low query traffic, in this paper we adopt a more standard approach by studying the comparative performance of a number of scheduling algorithms by assuming that queries are jobs to be placed in a set of machines. Namely we ignore the effect of communication and resource competition arising under high traffic of queries.

Search engines use inverted files as index data structures to speed-up the processing of 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 K documents as the query answer.

The amount of work required to solve a query is proportional to the sum of the lengths of the posting lists associated with the respective terms. Also queries are received by a broker machine which is in charge of routing them to the cluster processors. Thus the broker can roughly predict the running time of queries by using the length of the posting lists. We use this feature to let the broker schedule the queries onto the cluster processors. We also ignore communication costs since in our actual implementations of inverted files running on cluster of computers, we have observed that they are much less relevant with respect to the cost of query ranking. Thus in this paper we focus on algorithms devised to schedule ranking of queries efficiently.

2 Experiments

The results were obtained using a 12GB sample of the Chilean Web taken from the www.todocl.cl search engine. Queries were selected at random from a set of 250,000 queries taken from the todocl log. The experiments were performed on a cluster with dual processors (2.8 GHz). We used BSP, MPI and PVM realizations of the distributed inverted files. The scheduling algorithms were executed to generate the queries to be injected in each processor during the runs.

To evaluate the performance of the different scheduling algorithms we have used a set of metrics which we define in the following.

- Average Response Time (ART), is the average time elapsed between the query arrival and its departure from the search engine.
- Makespan (MS), is maximum load observed in any processor at a given instant of time.
- Current Makespan/Optimum (MAO), which is MS divided by the optimal job assignment (the average load considering all machines).
- Load Balancing (LB), is the MS divided by the machine with the least load.

We evaluated the performance of a number of so-called on-line scheduling algorithms since these algorithms are devised to process a stream of queries arriving, in our case, to the broker machine. That is, the decision of where to place a given query is taken at the time the queries arrives to the broker independently of the queries to arrive in the future. The algorithms we evaluated are (**A**) Albers [1], (**B**) Avidor et. al [2], (**C**) Bartal et. al [3], (**D**) Galambos et. al [10], (**E**) Graham [12], (**F**) Gomez [11], (**G**) Karger et. al [14], (**H**) Leischer et. al [9], and (**I**) Round-robin, that is, circular assignment of queries to processors.

We run the scheduling algorithms assuming different number P of processors (machines), namely P= 8, 32, 128, and 512.

In table 2 we show results for the average response time (ms) for the different algorithms and number P of processors. The results show that the algorithms by Albers (A), Graham (E) and Gomez (F) present the smaller response times respectively.

The figure 1 shows results for the efficiency metric MAO where values closer to 1 indicate better performance. For large number of processor the Graham (E) strategy presents a better and more stable performance than the other two strategies. This metric tell us which strategies are best suited to improve the overall throughput of the query processing. The load balance metric (LB) shows similar conclusions in figure 2. Namely the Graham algorithm keeps a better load balance than the others with the advantage that the response time is still competitive (about 20%). A key issue here is that the Albers' algorithm requires the manual setting of operational parameters whereas Graham's algorithm does not require any tuning.

We also for the Sync search engine in [18] we evaluated scheduling algorithms which are dynamic and static ones [13, 16, 5, 8, 6, 12]. We reproduce here the main results.

Algorithm	P=8	32	128	512
A	161,2	41,8	10,2	2,4
B	198,4	57,2	13,0	3,4
C	242,0	63,1	16,9	3,9
D	228,0	52,6	13,1	3,2
E	198,9	50,5	12,9	3,0
F	198,2	50,6	13,2	3,0
G	382,7	95,0	24,4	5,9
H	235,5	62,5	15,4	3,8
I	200.1	52.8	14.0	3.2

Table 1. Results for the average response time per query ranking

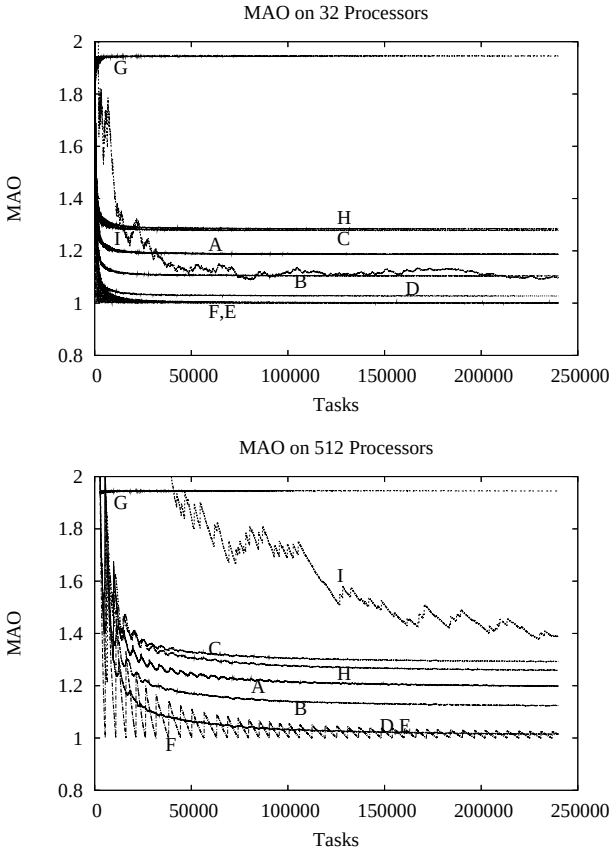

Figure 1. The MAO metric using 32 and 512 processors respectively.

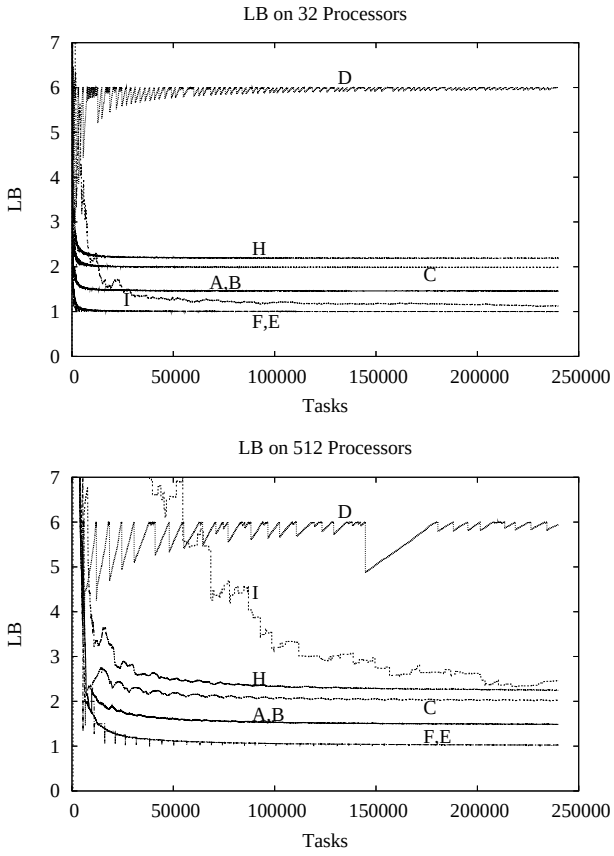

Figure 2. Performance for the Load Balance metric for 32 and 512 processors respectively.

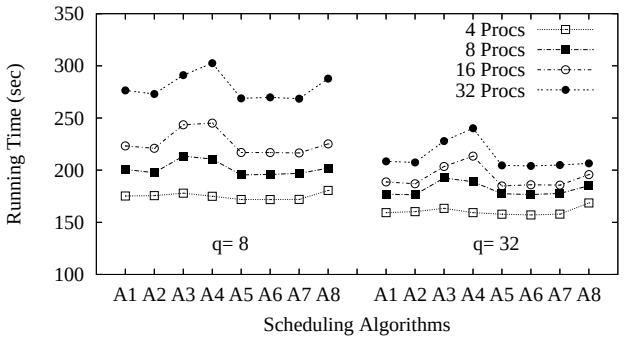

Figure 3. Total running time for 10,000 queries per processor using small queries under high and low traffic of queries.

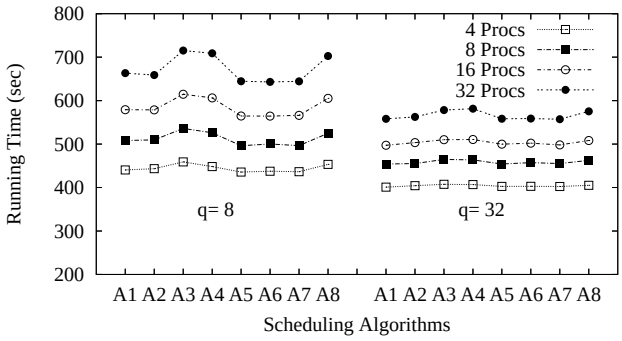

Figure 4. Total running time for 10,000 queries per processor using large queries under high and low traffic of queries.

Notice that in some cases we even gave them the advantage of exploring the complete query log used in our experiment in order to formulate a schedule of the queries. In others we allowed them to take batches of $q\,P$ queries to make the scheduling where q indicates the query traffic intensity. The algorithms evaluated are the following: (**A1**) Round-robin, namely distribute circularly the queries onto the processors (labeled **I** in the Async experiments), (**A2**) Graham algorithm (least loaded processor, labeled **E** in the Async experiments), (**A3**) Gomez algorithm (labeled **F** in the Async experiments, (**A4**) Optimal limit algorithm considering information in all processors, (**A5**) LPT (longest processing time) [7], (**A6**) FFD (first-fit decreasing) [4, 13, 15], (**A7**) BFD (best-fit decreasing) [4, 13, 15], and (**A8**) Alder's algorithm [1].

The following figures show running times for the bulk-synchronous realizations of distributed inverted files.

The figure 3 shows that the strategies A1, A2, A5, A6 and A7 achieve similar performance. A3 and A4 show poor performance. In most cases queries contain one or two terms. To see if the same holds for more terms per query we artificially increased the number of terms by using composite queries obtained by packing together several queries selected uniformly at random from the query log to achieve an average of 9 terms per query. This can represent a case in which queries are expanded by the search engine to include related terms such as synonyms. The results are presented in figures 4 which also shows that A1, A2, A5, A6 and A7 achieve similar performance. Among all the algorithms considered, A1 is the simplest to implement and its efficiency is outstanding. However, its performance in the Async case is not satisfactory, then we recommend using the Graham algorithm, namely **E** or **A2**.

3 Concluding remarks

We have presented an empirical evaluation of a number of general purpose job scheduling algorithms that can be used in the context of query processing in search engines. The most important feature of this application is that scheduling has to be done in a on-line manner as queries arrive to the broker machine. Another crucial issue is that the algorithm must be very efficient to avoid the broker become a bottleneck. Both in (standard) asynchronous search engines and synchronous search engines the classic Graham's algorithm happens to be a good choice for this application domain, both in response time of individual queries and overall query throughput of the search engine. This algorithm works well both in the asynchronous and synchronous modes of operation of the search engine.

The Graham's algorithm is extremely simple and fast, it just assigns the query ranking to the processor with the least load at the time instant in which the query is sent to processing onto the cluster processors. The broker can keep a table with the length of the posting lists associated with the query terms and use this as a measure of the amount of work required to process the query. In addition, the broker can keep a priority queue organized by the amount of work (posting lists lengths) assigned to each processor.

Acknowledgment: Partially funded by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.

References

[1] Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.

[2] Adi Avidor, Yossi Azar, and Jiri Sgall. Ancient and new algorithms for load balancing in the $_{\mathrm{p}}$ norm. *Algorithmica*, 29(3):422–441, 2001.

[3] Yair Bartal, Amos Fiat, Howard Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.

[4] J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, and L. A. McGeoch. Some unexpected expected behavior results for bin packing. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 279–288, New York, NY, USA, 1984. ACM.

[5] Jon Louis Bentley, David S. Johnson, Frank Thomson Leighton, Catherine C. McGeoch, and Lyle A. McGeoch. Some unexpected expected behavior results for bin packing. In *STOC*, pages 279–288, 1984.

[6] Onno J. Boxma. A probabilistic analysis of the lpt scheduling rule. In *Performance*, pages 475–490, 1984.

[7] Onno J. Boxma. A probabilistic analysis of the lpt scheduling rule. In *Performance '84: Proceedings of the Tenth International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 475–490, Amsterdam, The Netherlands, The Netherlands, 1985. North-Holland Publishing Co.

[8] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. *Approximation algorithms (ed. D. Hochbaum)*, chapter Approximation algorithms for bin packing - a survey. PWS, 1997.

[9] Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *European Symposium on Algorithms*, pages 202–210, 2000.

[10] Gábor Galambos and Gerhard J. Woeginger. An online scheduling heuristic with better worst case ratio than graham's list scheduling. *SIAM J. Comput.*, 22(2):349–355, 1993.

[11] Carlos Gómez. Gestión de brokers en bases de datos paralelas y distribuidas. Master's thesis, Universidad de Santiago de Chile, November 2004.

[12] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[13] David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, M. R. Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3(4):299–325, 1974.

[14] David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20(2):400–430, 1996.

[15] F T Leighton and P Shor. Tight bounds for minimax grid matching, with applications to the average case analysis of algorithms. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 91–103, New York, NY, USA, 1986. ACM.

[16] Frank Thomson Leighton and Peter W. Shor. Tight bounds for minimax grid matching, with applications to the average case analysis of algorithms. In *STOC*, pages 91–103, 1986.

[17] M. Marin and V. Gil-Costa. $(\mathrm{Sync}|\mathrm{Async})^{+}$ MPI Search Engines. In *PVM/MPI*, pages 117–124, Oct. 2007. Lecture Notes in Computer Science 4757.

[18] Mauricio Marin and Carlos Gomez. Load balancing distributed inverted files. In *WIDM '07: Proceedings of the 9th annual ACM international workshop on Web information and data management*, pages 57–64, New York, NY, USA, 2007. ACM.

[19] L.G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33:103–111, Aug. 1990.