Exploring Load Balancing in Parallel Processing of Recursive Queries *

Sérgio Lifschitz[1]**, Alexandre Plastino[2]*** and Celso C. Ribeiro[1]†

[1] Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brasil
[2] Universidade Federal Fluminense, Departamento de Ciência da Computação
Praça do Valonguinho s/n, Niterói RJ 24210-130, Brasil

Abstract. We study a *dose-driven* dynamic load balancing strategy to evaluate database recursive queries. This proposal aims at obtaining a good workload balance with full use of the available resources, with different kinds of skew considered. The strategy is intended for general recursive queries and preliminary computational results illustrate its efficiency when applied to the particular case of linear transitive closure.

1 Introduction

Recent work on load balancing, mostly on join processing, has confirmed its importance when one wants to achieve good performances during the evaluation of parallel database queries. We are interested here in more complex queries, such as the recursive ones. In this case, the work due to a task cannot be previously determined and, consequently, no method can define at the outset the tasks to be executed in parallel in order to balance the workload at each processor.

We claim that the set of tasks to be executed should be assigned to each site dynamically during the query evaluation process, with any workload imbalance controlled, avoided if possible. This way, we may take into account different kinds of skew: intrinsic (related to the input data distribution), partition (due to the algorithm to be performed or to the parallel strategy considered) [WDJ91] and also, what we call here a concurrency skew, due to either a multi-user (or multi-transaction) or an heterogeneous processing environment.

The main contributions of these paper are: (1) the introduction of a dynamic load balancing strategy for processing recursive queries, that deals well with different skew reasons and keep all processing sites active most of the time and (2) an initial implementation of the strategy in a parallel environment, here applied to the simple, yet important, case of linear transitive closure queries.

The paper is organized as follows. In Section 2, the general strategy is explained and some related work is presented. Next, in Section 3, its specialization

* Work partially done in LNCC, Laboratório Nacional de Computação Científica.
** E-mail: lifschit@inf.puc-rio.br. Supported in part by CNPq under grant 300048/94-7.
*** E-mail: plastino@dcc.uff.br.
† E-mail: celso@inf.puc-rio.br. Supported in part by CNPq under grant 302281/85-1.

when applied to the transitive closure query is discussed, with preliminary experimental results also given. Conclusions and future work are found in Section 4.

2 Dose Driven strategy

We present a dynamic task-oriented demand-driven parallel processing strategy for dealing primarily with recursive queries. It is dynamic, in the sense that the workload is assigned to each site during the evaluation process, and demand-driven because new tasks are allocated to a site when it becomes idle and asks for new tasks to process. Our focus is on determining and controlling the execution when tasks are being distributed, in order to avoid load imbalance at most, rather than correcting it later whenever it occurs. Tasks may have variable sizes so to better tune which and how many tasks are to be run at each site. Thus, our strategy is called *dose-driven*, which stands for a method that aims at obtaining an even workload distribution with variable-sized tasks corresponding to doses that are dinamically assigned to the parallel sites.

A task-oriented demand-driven strategy has been suggested previously in [LT92] for balancing the load during the processing of join queries. These are clearly less complex than recursive ones and the strategy proposed is guided by a specific algorithm (hash join), which limits the way tasks are defined. Also, we consider load balancing as part of the whole strategy and not only for redistribution purposes when an uneven workload assigned to parallel sites is detected.

In the case of recursive queries, existing works appear in the context of (datalog) rule programs, where the general framework for processing recursive queries is known as *data reduction* (or *rule instantiations* paradigm [WO93]. The main idea is to parallelize the query evaluation by assigning subsets of the rule instantiations (which are obtained by appending arithmetic predicates like hash functions to some of the rules) among the sites, such that each site evaluates the same program but with less data.

A load balancing method is proposed in [WO93] where a list of alternative parallelization strategies (a set of different restricting predicates) could be used to change the strategy dinamically whenever a workload imbalance is detected. However, a better performance cannot be guaranteed with the new strategy chosen and there is no simple way to implement this mechanism. In [DSH+94], a more sophisticated parallel strategy is proposed, which includes a *predictive* protocol for detecting potential uneven processing at each site and a *correction* algorithm that balances the load. The problem here is that some considerations about load imbalancing that may not be true in practice are made. Indeed, it is considered that a larger local database in a given site implies more work in the future and this is not always the case, as for the join product skew. Moreover, the concurrency skew is not considered.

In our dose driven strategy, we keep the rule instantiations partitioning method but each site is able to evaluate any of the restricted versions and not only those assigned initially. In our case, each subset of instantiations is considered a task. There are many ways to determine these tasks and the strategy will define which site will execute a given task. Basically, these could be subsets of constants in the active domain of the database, or generated by horizontal and/or vertical partitioning of database relations. As so, we have all the flexibility needed to guide the evaluation strategy and control uneven workload in different skew conditions.

In order to illustrate the applicability of our proposed strategy and its tasks generation method, we have adapted it to the case of the transitive closure query. We compare the behavior of our approach to that proposed in [AJ88] - called here AJ - which is a static distribution strategy that does not include load balancing techniques and corresponds to the data reduction paradigm applied to the linear transitive closure query. Further details can be found in the full paper [LPR96].

3 Experimental Results

In this section, we give the specialization of our strategy for the evaluation of the transitive closure of a binary relation, say R, usually defined as follows:

$$r_1 : Tc(x, y) :- R(x, z), Tc(z, y).$$
$$r_2 : Tc(x, y) :- R(x, y).$$

The evaluation of the Tc relation may be understood as the computation of all successors of all nodes in the relation's corresponding direct graph. Thus, we may define the tasks to be executed as the computation of all successors of a subset of constants in R. In its linear definition, as above, the transitive closure evaluation can be executed in parallel with no communication during the evaluation process (known as *pure parallelization*), as long as R is replicated through all sites. This property of recursive programs is called *decomposability* [WO93].

The number of tasks t that will be generated must be bigger than p, the number of processing sites available. We could determine each of the tasks as follows: considering an order in the constants set taken into account by the query, a task will be the pair (i, j), where i is the i-th constant in the domain and j represents the number of constants belonging to the task itself.

One of the sites, the coordinator, controls the distribution of tasks to all p sites. There are two phases: in the first one, the coordinator distributes p tasks, exactly one allocated to each site. In the second and last phase, it is time for the dynamic and adaptive distribution of the remaining $t - p$ tasks. As long as there are still tasks to be distributed, the coordinator site sends a new task to a site upon request as soon as it becomes idle. When all tasks have been assigned to the processing sites, the evaluation has reached its end.

We have made preliminary implementations for both methods DD and AJ, so to investigate their behavior in different skew situations. We have done all implementations on the IBM 9076 SP/2 machine. All programs were coded in C/SQL (with a complete Open Ingres DBMS available at each node). To make full use of the parallel environment, MPI (Message Passing Interface) was chosen.

In order to better illustrate our experiments, we have chosen two input binary relations among all those randomly generated. Relation R1 has 5,000 tuples

(150,000 in the closure) and R2 10,000 tuples (1,000,000 in the closure). Both R1 and R2 correspond to 1,000 nodes graphs, R1 being acyclic, R2 cyclic. In particular, we see that the closure of R2 is the 1,000 nodes complete graph.

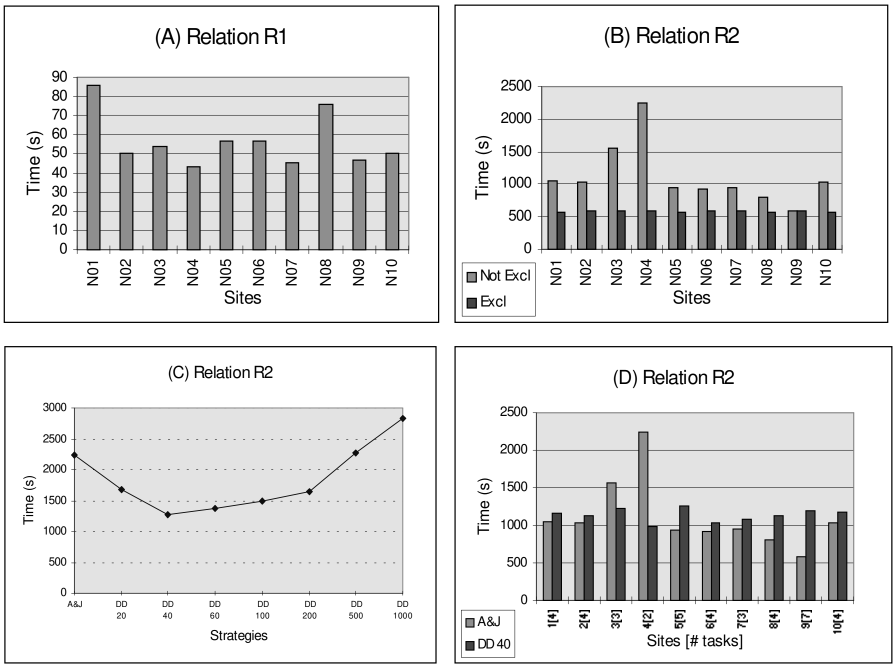

Fig. 1. (A) *AJ* for R1: exclusive environment (B) *AJ* for R2: exclusive and non-exclusive environments (C) *DD* parallel time for R2 and (D) *DD* distribution of 40 tasks

To illustrate the effects of partition skew, here related to the unknown workload associated to each task assigned to a processing node, we show in Figure 1(a) bar coded graphics representing the evaluation of the transitive closure query by the *AJ* strategy in a single-user (exclusive) parallel environment. Times are given in seconds and N01, N02, ... N10 are the 10 sites used. As we can see, there is a strong workload imbalance. For example, site N01 has taken twice as much the time N04 has processed its job.

We consider now a non-exclusive environment where there are other processes - accessing the database or not - running concurrently on the same sites. Figure 1(b) shows *AJ* algorithm on relation R2 in this situation. An uneven processing time has occurred, although the work at each site was equivalent.

We have also tried out the *DD* strategy with distinct total number of tasks, ranging from 20 up to 1000 (one task corresponding to exactly one attribute constant) tasks. In Figure 1(b), it can be seen that the parallel time of the *AJ* strategy for relation R2 is 2240 seconds and in Figure 1(c), with 20 and 40 tasks to be dynamically allocated to the sites, *DD* has obtained parallel processing times of 1676 and 1263 seconds respectively. One could think that a continuous increase on the number of tasks would imply even better results. However, as shown in Figure 1(c), as the number of tasks increase, the parallel time keep its ascending curve, where it gets even worse than the AJ algorithm. The problem is that the query processing work, when partitioned in a set of tasks, has a fixed cost per task that is intrinsically sequential.

In Figure 1(d), we observe the actual distribution of tasks that occurred for algorithm DD with 40 tasks, which has obtained the best parallel time before. In the horizontal axis, *I[J]* indicates that site *NI* has performed J tasks. So, we see that N04 has executed only 2 tasks, as its external load was high, while site N09 was responsible for 7 tasks, almost 20% of the total number of tasks. There is not only a gain in the efficiency of the parallel query processing but also it is shown that a good workload balancing was obtained.

4 Final Comments and Future Work

There are many interesting points to discuss and further explore. First, we believe that an increasing number of tasks is valid while the sum of fixed costs related to every task does not offset the gain in performance obtained by the DD strategy. It is still an open question if there is any fixed cost variation with respect to the size of the tasks and this must be better investigated.

Some other results that we have obtained [LPR96] point out that, if the minimization of the total parallel time is the goal to be achieved, not always the best workload balance corresponds to the best parallel strategy. It deserves further investigation but so far we believe that these results are due to the fixed cost mentioned above. We will also implement variable-sized tasks during the evaluation process, probably with a non-blind strategy in mind, that enable a fine tuning of tasks sizes.

References

[AJ88] R. Agrawal and H.V. Jagadish "Multiprocessor Transitive Closure Algorithms" *Procs. Intl. Symp. Database in Parallel and Distributed Systems*, 1988, pp 56–66.

[WDJ91] C.B. Walton, A.G. Dale and R.M. Jenevein "A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins", *Procs VLDB*, 1991, pp 537-548.

[DSH+94] H.M. Dewan, S.J. Stolfo, M.A. Hernandez and J-J. Hwang, "Predictive Dynamic Load Balancing of Parallel and Distributed Rule and Query Processing", *Procs. ACM-SIGMOD Intl. Conf. on Management of Data*, 1994, pp 277-288.

[LT92] H. Lu and K-L. Tan, "Dynamic and Load-Balanced Task-Oriented Database Query Processing in Parallel Systems", *Procs. Intl. Conf. on Extending Data Base Technology*, 1992, pp 357–372.

[LPR96] S. Lifschitz, A. Plastino and C.C. Ribeiro, "Exploring Load Balancing in Parallel Processing of Recursive Queries" *Technical Report MCC37, PUC-Rio, Departamento de Informática*, November 1996.

[WO93] O. Wolfson and A. Ozeri, "Parallel and Distributed Processing of Rules by Data-reduction", *IEEE Trans. Knowledge and Data Eng.* 5(3), 1993, pp 523–530.