Offering Parallelism to a Sequential Database Management System on a Network of Workstations Using PVM

Mathieu Exbrayat[1] and Harald Kosch[2]

[1] Laboratoire d'Ingénierie des Systèmes d'Information – Institut National des Sciences Appliquées de Lyon
F - 69621 Villeurbanne Cedex
[2] Laboratoire d'Informatique du Parallélisme – Ecole Normale Supérieure de Lyon
F - 69364 Lyon Cedex 07.

Abstract. The considerable growth of on-line document searching and consulting brings much of the data providers to reconsider their database management systems (DBMS) capacities. Parallel DBMS then appear as a good solution, but the involved changes in administration and cost limit their breakthrough. To overcome these drawbacks, we propose an hybrid structure, which adapts a parallel extension to an existing DBMS. This extension cuts down the amount of work of the sequential DBMS, by parallelizing the incoming queries over a network of workstations communicating with PVM.

Keywords : Parallelism, Networks of Workstations, Relational Databases.

1 Introduction

The last ten years have witnessed the arising of parallel techniques into Database Management Systems (DBMS), in order to provide quick access to large and very large databases to many simultaneous users. Two domains are especially concerned : Online Transaction Processing (OLTP, i.e. business databases) and Query Processing (QP, i.e. data extraction). OLTP deals with fast and reliable updates, as it involves unduplicatable means, such as money, or plane tickets, while query processing deals with high bandwidth and wide storage, as it is employed on Decision Support Systems. Speed of updates, and time in general, are less significant here, as insconsistencies appear slower, and are generally less dangerous.

Many studies have been carried out in these contexts. The main topics are data fragmentation [1,2], Parallel Execution Plans [3,4], and duplication strategies [5]. Most of the work done was designed for Massively Parallel Machines (MPM), which are powerful but quite expensive. This made hybrid architectures, such as workstation clusters, or networks of workstations come to the front page of research [6,7]. More than suggestive aspects, such as global cost, it can be considered that workstations are widely used by many companies. They

provide a satisfying robust and extensible computing power compared to most parallel machines on the market.

This allows us to believe that virtual parallelism between small- to middle-size computers is a promising domain of investigation in the database area. In this paper, we consider a different approach of parallel databases, which is oriented towards Query Processing (and more specifically toward document databases), and based on an original architecture. Our goal is not to build another parallel DBMS, but to create an extension, connected to an existing sequential DBMS. This means that we don't look at terabyte databases, but at overloaded DBMS which manage some gigabytes or tens of gigabytes databases. Considering resource management abilities of parallel libraries, we decided to use PVM, as it allows us to keep control over execution sites, and as it appears to be more adapted to heterogeneous systems than other products.

We present in section 2 our DBMS structure. Section 3 describes how we adapt SQL queries to the structure. In Section 4 we detail the data distribution. In section 5 we compare our proposal with related work.

2 Description of the extended DBMS

Our system has four main components (see fig. 1). The first two are the clients and the existing DBMS (EDBMS). This latter is not modified, and the only modification that clients must perform is their connection point. The other two components are specific to our system. They are i) the so-called *server*, which interfaces the EDBMS with its clients, and allows a parallel execution by catching and transforming queries into a parallel form, and ii) the so-called *calculators*, which run on the nodes (workstations) of the LAN. Each *calculator* stores database fragments, and executes queries over these fragments. Communications between the *server* and the *calculators* (and between *calculators* when they exchange data) are done using PVM [8]. While systems as in [9] propose to get multiple virtual processors over each node, we propose to get only one *calculator* per station. This is driven by the fact that this limits the volume of the fragmentation dictionary, and then allows a lighter distribution management. On another side, we are currently implementing a multi-threaded release of our prototype, which will allow several simultaneous queries on a single calculator.

3 Catching and parallelizing queries

Queries are caught by changing the connecting point of applications to our extension. This allows us to examine each query in order to parallelize it when it seems possible. With no parallelism needed, the SQL query is sent to the EDBMS, then results are returned to the it server (see fig. 2) and back to the client. In case of parallelization, the parallel execution plan, or PEP (i.e. an ordered set of elementary instructions representing the query global execution) is transformed into execution parameters destinated to the *calculators*. Each of these contains both execution and re-fragmentation information. This facilitates coarse-grained

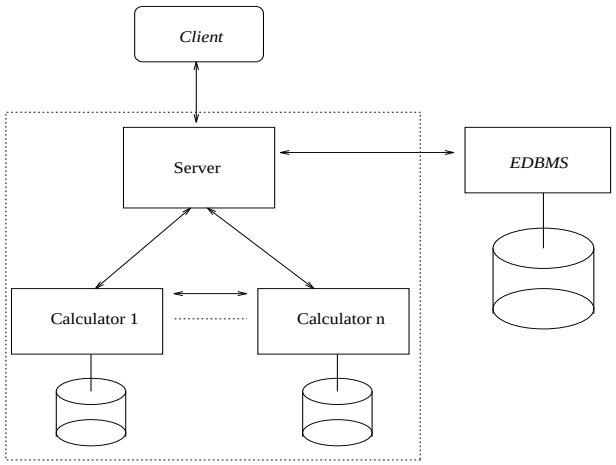

Fig. 1. The extended DBMS Structure

(bucket by bucket) pipelined transmission of matching tuples within a single instruction. The instruction structure is common to unary (selection, projection) and binary (such as joins) operations, in order to directly put the PEP elements into instructions and send them.

4 Distribution of data

4.1 Calculators

Machines where *calculators* should be placed are chosen according to their level of use, which is determined among disk availability, memory availability, disk accesses and cpu use [10,11]. During use, availability is controlled by regularly scanning the workstations' level of use. To be more precise, *calculators* are composed of three elements (see fig. 3): i) an *interface*, that gets data and instructions, and sends results to the *server* or to another *calculator* (intermediate results). Under this *interface*, we have two elements, ii) a *storage unit* and iii) a *query execution unit*, which communicate through message passing and shared memory.

4.2 Data elements

Each fragment is divided into buckets. These are transfer-oriented buckets, in the way that their size depends on transfer efficiency criteria rather than available memory criteria. Despite the machines diversity, we have chosen to use physically equal-sized buckets, as it allows a good effectiveness of sub-queries (buckets are considered as the atomic data transfer volume). Bucket is the parallelism grain, and load-balancing is then made possible with a quick control

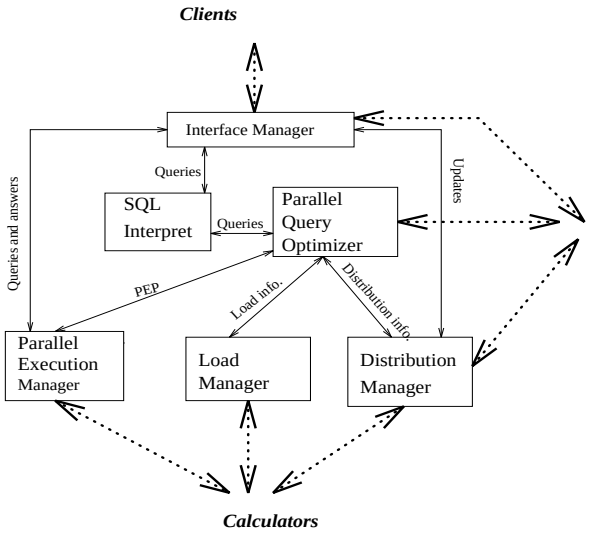

Fig. 2. The modules of the server

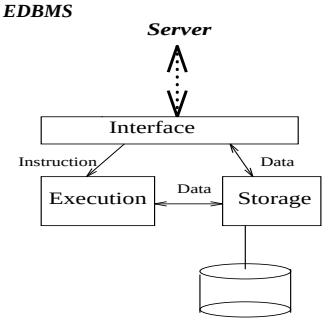

Fig. 3. The modules of a calculator

over the number of already tested buckets. Let us describe how we choose the bucket size in a concrete situation. This example is conducted over two sparc 5 workstations, both connected to Ethernet and ATM networks. Communication between machines is assured by PVM. We use a ping-pong algorithm to get a "pack+transfer+unpack" time. On another side, we used a small algorithm derived from the first one, which gives a "pack+unpack" time. Packing is done by using "pvm_pkstr" with strings of different lengths. Merging the two results, we obtained the transfer time, the visible throughput (in a point-to-point transfer) of the LAN (see fig. 5), and the percent of global transfer time spent in packing and unpacking data (see fig. 4).

Concerning the throughput, we notice that the optimal size of bucket is greater or equal to 3 kilobytes. Beyond this size, the LAN throughput stays quite stable at about 1 Megabyte per second for the ATM network, and 0.6 Megabyte per second for the Ethernet network. We interpret this low values as the result of low-level over-costs. They just represent the point-to-point throughput, and not the maximal capacity of the LAN. Looking at fig. 4, we notice that the "packing percentage" grows with the bucket size from 2 percent (size=512 kB) to 8 to 10 percent when the size is over 30 kB. The optimal bucket size is then over 3 kB. To decide on a satisfying size, we then must consider the fact that big buckets will limit the efficiency of pipelining between calculators. (The smaller the bucket, the more efficient the pipelining). As throughput and package are quite stable over 10 kB. Then we can adopt a 10 kB bucket size.

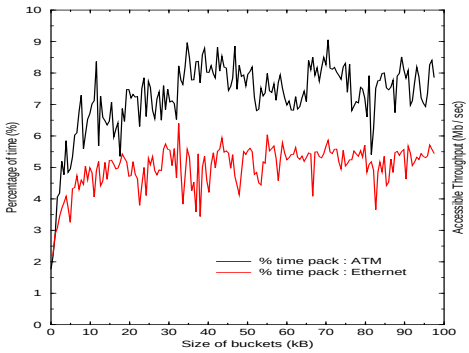

Fig. 4. Percentage of global transfer time spent in packing and unpacking data

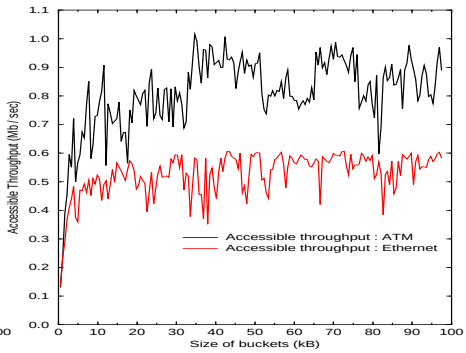

Fig. 5. Observed throughput in a point-to-point data transfer

4.3 Description of distribution tasks

Original distribution is done at launch time (before users' connections are allowed). The *distribution manager* extracts and sends data on all available *calculators*. Those are then asked to execute fragmentation queries. Execution is divided in two steps, i) select all tuples in the concerned fragment and ii) fragment these pseudo-results according to the definitive fragmentation criteria.

In case of a transaction, the corresponding query is directly sent to the EDBMS. In case of commit, the *interface manager* ask the *distribution manager* to check if distributed data have become out of date. Added, modified, or suppressed tuples are then communicated to the *Distribution Manager*, which informs the concerned *calculators* that they must add, change or suppress one or more tuples. In case of suppression, concerned buckets only become lighter. In case of addition, a new bucket can be created if all existing ones are full.

5 Related work

In the last three years several research work began to implement a parallel DBMS on top of a network of workstations using PVM [12–14]. Classically, two approaches are presented in the context of the implementations. In a first approach, the parallel database is implemented from scratch [14]. In the second, implementation is done by parallelizing an existing sequential DBMS. [12,13]. The latter profits of software reuse, as the first could exploit more parallelism.

However the amount of work needed to achieve parallel functionalities remains important. For example, the implementation of Volcano [13] took about five years. Given the present context of concurrency and users' demand, the industry cannot afford such long development delays. Furthermore, most of the networks of workstations are not only dedicated to parallel databases, many users should be provided with many applications. In such a scope we integrated our parallel extension as a whole component of the existing DBMS and make

only use of the parallel extension if enough resources are available on the network of workstation and if the existing DBMS is overloaded. Interfacing is enabled by means of the available input/output functions of this DBMS. Development costs are decreased and security functionalities, such as backup and data recovering, are easily maintained.

6 Conclusion and future work

In this paper we described the architecture of a parallel extension for a sequential and relational database management system built on a network of workstations using PVM. While previous work interested mainly in implementing a fully functional parallel DBMS on top of the network of workstations, we are interested in cutting down the work of an overloaded database server,to be executed on available resources of a local network of workstations. This saves considerably development costs and integrate the full functionality of the sequential DBMS. Special emphasis was put on the definition of the appropriate bucket sizes for minimizing the communication costs.

We can conclude by underlining the fact that our system, while originally oriented toward overloaded DBMS, could be used in other contexts. By example, it could easily be turned into a federated DBMS: we can select interesting data from several databases, managed by several DBMS, get them to our local area network, and exploit them intensively with no direct connection to their original site.

References

1. D. DeWitt and J. Gray, "Parallel Database Systems : The Future of High Performance Database Systems," *Communications of the ACM*, vol. 35, pp. 85–98, June 1992.
2. R. Gallersdörfer and M. Nicola, "Improving Performance in Replicated Databases through Relaxed Coherency.," in *Proceedings of the 21st VLDB Conference*, (Zurich, Switzerland), 1995.
3. D. Schneider and D. DeWitt, "A Performance Evaluation of Four Parallel Algorithms in a Shared-Nothing Multiprocessor Environment," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (Portland, Oregon, USA), June 1989.
4. W. Hasan and R. Motwani, "Coloring Away Communication in Parallel Query Optimization," in *Proceedings of the 21st VLDB Conference*, (Zurich, Switzerland), pp. 36–47, 1995.
5. D. Chamberlin and F. Schmuck, "Dynamic Data Distribution (D3) in a Shared-Nothing Multiprocessor Data Store," in *Proceedings of the 18th VLDB Conference*, (Vancouver, British Columbia, Canada), 1992.
6. L. Chen, D. Rotem, and S. Seshadri, "Declustering Databases on Heterogeneous Disk Systems.," in *Proceedings of the 21st VLDB Conference*, (Zurich, Switzerland), 1995.
7. X. Zhang and Y. Song, *The State-of-the-art in Performance Modeling and Simulation : Computer and Communication Networks*, ch. 4, An integrated approach of performance prediction on networks of workstations. K. Bagchi, J. Walrand and G. Zobrist, Eds, Gordon and Breach, 1996.
8. M. G. et al., *PVM : Parallel Virtual Machine*. Cambridge, USA: MIT Press, 1994.

9. D. Schneider, D. DeWitt, J. Naughton, and S. Seshardi, "Practical Skew Handling in Parallel Joins," in *Proceedings of the 18th VLDB Conference*, (Vancouver, British Columbia), Aug. 1992.
10. F. Douglis and J. Ousterhout, "Transparent Process Migration : Design Alternatives and the Sprite Implementation," *Software - Practice and Experience*, vol. 21, pp. 757–785, Aug. 1991.
11. M. Mutka and M. Livny, "The Available Capacity of a Privately Owned Workstation Environment," *Performance Evaluation*, vol. 12, pp. 269–284, July 1991.
12. G. Bozas, M. Jaedicke, A. Listl, B. Mitschang, A. Reiser, and S. Zimmermann, "On Transforming a Sequential SQL DBMS into a Parallel One : First Results and Experiences of the MIDAS Project," in *EUROPAR'96* (Springer, ed.), no. 1124 in LLNCS, pp. 881–887, Aug. 1996.
13. G. Graefe and D. L. Davison, "Encapsulation of parallelism and architecture independance in extensible database query processing," *IEEE Transactions on Software Engineering*, vol. 19, July 1993.
14. N. Papakostas, G. Papakonstantinou, and P. Tsanakas, "PPARDB/PVM : A Portable PVM Based Parallel Database Management System," in *ACPC 96 Conference Series* (S. Verlag, ed.), vol. LNCS 1127, 1996.