A Parallel Extension for Existing Relational Database Management Systems

Matthieu Exbrayat
LISI- INSA Lyon
20, av. A. Einstein
F-69621 Villeurbanne Cedex
Tel : +33 04-72-43-89-83
Fax : +33 04-72-43-85-97
E-mail : exbrayat@lisiflory.insa-lyon.fr

Harald Kosch
LIP - ENS Lyon
16, allée d'Italie
F-69361 Lyon Cedex 07
Tel : +33 04-72-72-85-03
Fax : +33 04-72-72-80-80
E-mail : hkosch@lip.ens-lyon.fr

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. To allow this parallel execution, data are replicated over the stations according to a specific strategy. We describe this strategy and then examine the parallel query optimization.

***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), both of which have specific needs.

OLTP deals with fast and reliable updates, as it involves unduplicatable means, such as money, raw materials, 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 Speed of updates, and time in general, are less significant here, as the kind of information is either getting slowly out of date (e.g. books lists), or bringing few considerable incoherence (e.g. statistical studies data).

Many studies have been carried out in the context of OLTP or QP. The main topics are data fragmentation [1,2], Parallel Execution Plans [3,4], and duplication strategies [5].

Most of the work done was built on the assumption that parallel DBMS would run on Massively Parallel Machines (MPM). Such machines, while causing an incomparable rise of performance, are still quite few, and represent a big investment. 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 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. The system we designed appears to be hybrid in at least two ways: firstly by using a network of workstation, secondly by recycling and integrating an existing DBMS.

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 the query execution parallelizer is presented, and in section 6 we compare our proposal with related work.

2. Description of the extended DBMS

Our system has four main components (see Figure 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. While systems as in [8] 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 dictionnary, and then allows a lighter distribution management.

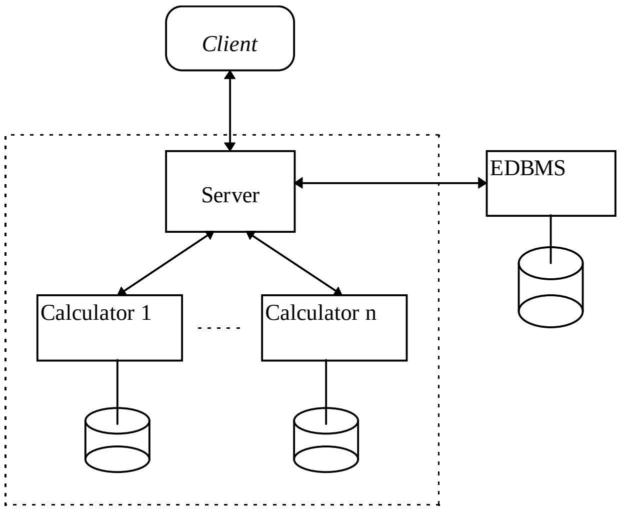

Figure 1 : The extended DBMS Struture

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 (see section 0). With no parallelism needed, the SQL query is sent to the EDBMS, then results are returned to the *server* (see Figure 2) and back to the client. No parallelism is achieved whether if some concerned tables have not been distributed, or if the EDBMS level of use is slightly light, or if the "query" is a transaction. In case of parallelization, the parallel execution plan, or PEP (see section 5) is transformed into execution parameters destinated to the *calculators*. Each of these contains both execution and re-fragmentation information. This facilitates coarse-grained (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.

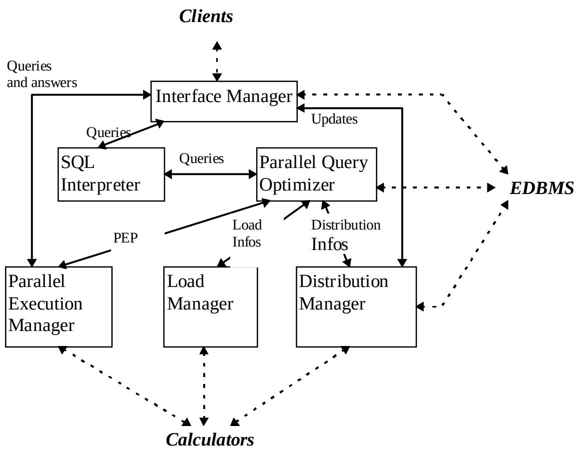

Figure 2 : The modules of the server

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 [9,10].

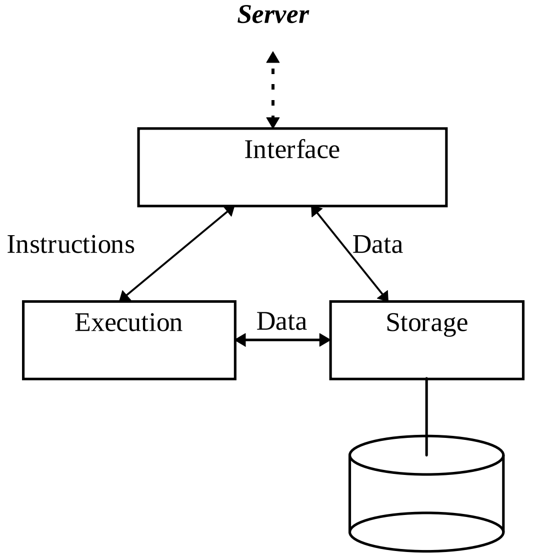

Figure 3 : The modules of a calculator

During use, availability is controlled by regularly scanning the workstations' level of use. To be more precise, *calculators* are composed of three elements (see Figure 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. First of all, this limits data transfer preparation as buckets do not need to be re-sized. This then allows a good effectiveness of sub-queries, as 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 over the number of already tested buckets. As a counterpoint, we can possibly meet quite empty buckets resulting of an intermediate operation. Buckets size is important as it influences the number of messages going accross the LAN. To sum up, buckets must be big enough to limit transfers, and small enough to be generally full. Let us describe how we choose the bucket size in a concrete situation.

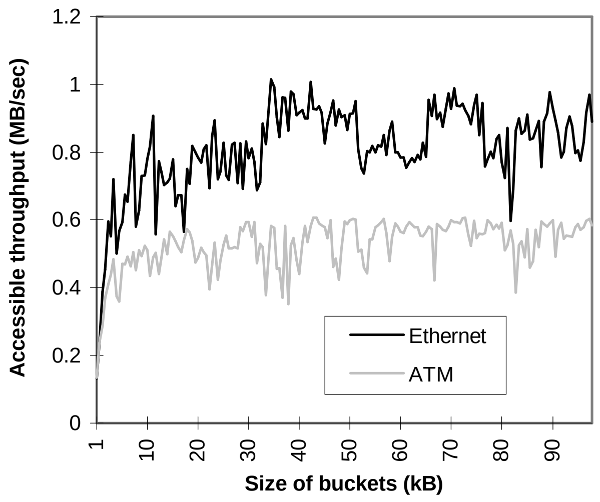

Figure 4 : Observed throughput in a point-to-point data transfer

This example is conducted over two sparc 5 workstations, both connected to ethernet and ATM networks. Communication beetween 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. Merging the two results, we obtained the transfer time, the visible throughput (in a point-to-point transfer) of the LAN (see Figure 4), and the percent of global transfer time spent in packing and unpacking data (see Figure 5).

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 low-level overcosts. They just traduce the point-to-point throughput, and not the maximal capacity of the LAN. Looking at Figure 5, 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 between 3 kB and 30 kB. We loose throughput under 3 kB, and over 30 kB the packing time becomes too important compared to the transfer time. As a satisfying size, we can use 20 kilobyte buckets.

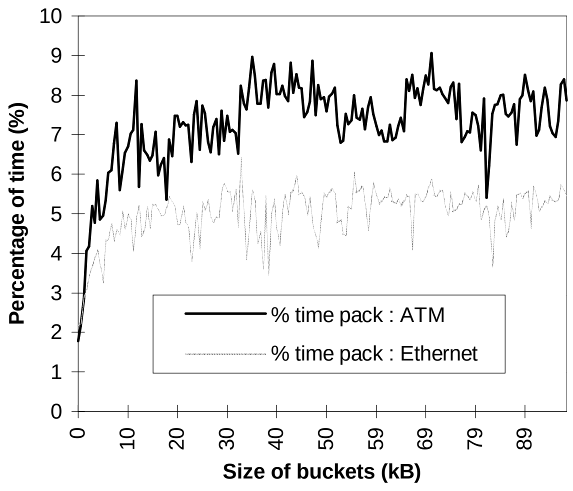

Figure 5 : Percentage of global transfer time spent in packing and unpacking data

4.3 Distribution strategy

Some factors can limit the global space available for data distribution.We can be faced with a lack of resources in a dedicated environment, or a user-limited availability in a non-dedicated environment, both in terms of disk space (direct impact) and in terms of cpu-time, that is to say in the amount of data that can be processed, and so stored, on one site (indirect impact).In this second case we are driven to such considerations as the "social contract" described by [11], that is to say, how much we can interfer with the workstations' own work.

When meeting these limits, we must apply a partial distribution strategy. We propose the following heuristic, based on queries frequency statistics, which can either be statically obtained from the treatment model, in case of a transaction processing application, or dynamically, according to real use, in case of consultation applications. We consider attributes frequencies rather than tables frequencies as fragmentation is done according to attributes.

Let us define *Fu* the frequency with which each attribute of each relation is accessed for a single unary query (i.e. select, project). We must then define *Fc*, the frequency for multiple terms query (joins, select+joins...).

In the first step of our algorithm we decide wether a relation has to be distributed or not. This phase can be divided in two sub-steps:

1.We eliminate tables where no attribute has a sufficient *Fc* regarding the mean value among all attributes. Let us call *Dc* the list of kept relations.

2.We can anyway choose to keep some relations eliminated in 1) where one or several of the most oftenly used selection attributes (i.e. looking at their *Fu*) are no indexed attributes. Let us call *Du* the list of the relations kept in this step.

Agressivity of this heuristic can be adapted in point 1 depending on the available volume and in point 2 when multi-terms queries are favoured.

In a second step, we can cut some attributes from the distributed tables, making a kind of vertical fragmentation. To choose between attributes, we must once again have a look at their frequencies:

1.Concerning tables in *Dc*, we can eliminate attributes which have a small *Fc* and do not participate in joins (we must note that this can arise after supressing tables in the first step).

2.Concerning tables in *Du*, we just have to look at *Fu*, as these tables were not kept according to their joins frequency.

We must limit the agressivity of this second step, as each attribute suppression tends to limit the number of parallelizable queries.

If data volume remained too high, we should have to apply the heuristic once or more than once with an increased agressivity in the first step.

Let us describe an exemple of the use of our heuristic. To make understanding easier, keys are marked with a star, and joins attributes have the same name. In the first step (see Table 1), the mean Fc value FcM is 11. We notice that for relations R4 and R5, all attributes have low Fc comparing with FcM. These two relations should be eliminated. But, having a look at R5, we notice that *Fu*(A8) > *Fu*(A2*) . We then decide to keep R5.

Relations	Attributes	Fc	Fu
R1	A1*	15	8
	A2	10	6
	A3	30	14
R2	A4*	25	3
	A5	7	8
	A1	12	15
R3	A6*	13	3
	A7	11	5
	A4	5	8
R4	*A2**	*3*	*6*
	A8	*8*	**14**
R5	*A9**	*4*	*7*
	A4	*3*	*4*

Table 1: Global Database Scheme

In the second step (see Table 2), FcM is 12, and both Fc(R2.A5) and Fc(R3.A4) are very low compared to it. We can then suppress these two attributes. Looking at Du, Fu(R4.A2*) is low comparing with Fu(R4.A8). We then decide to suppress R4.A2*.

The reduced scheme we obtain is presented in Table 3.

Relations	Attributes	Fc	Fu
R1	A1*	15	8
	A2	10	6
	A3	**30**	14
R2	**A4***	**25**	3
	A5	*7*	8
	A1	12	15
R3	A6*	13	3
	A7	11	5
	A4	*5*	8
R4	*A2**	-	*6*
	A8	-	**14**

Table 2 : Simplified database schema after tables suppression

Relations	R1			R2		R3		R4
Attributes	A1*	A2	A3	A4*	A1	A6*	A7	A8

Table 3 : Simplified database schema after attributes suppression

4.4 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. This means that data coherency and consistency is guaranted by 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. We must notice that our system is dedicated to Query Processing. Temporary inconsistency can appear beetwen data stored on the EDBMS and distributed data, due to the fact that a updates can be delaied on the stations. As coherency must be kept during updates, the current version of our prototype locks the whole distributed data during updates. This method is rather slow, and restricts most of updates to slack periods. This points out the fact that our approach cannot, until now, support intensive transaction management, but well fits for document search or decision supports, as this domains can tolerate delaied updates.

Some changes in the hardware (fluctuation of machines availability, machines shutdown) or in the database (deep changes in queries frequencies, numerous updates bringing fragmentation skews) can bring the need of a partial or total redistribution. Redistribution techniques have long been discussed [12,8,13] and do not need to be rediscussed here. The important point is how to get data to their new site. As long as database changes do not impose modifications in the list of distributed tables, redistribution is done by sending fragmentation queries as described above. When hardware changes are encountered, we must access the EDBMS to re-extract concerned data. In case of range partitionning, we select and extract data from the EDBMS where values of the partitionning attribute are between the two bounds of the lost fragment. In case of hashing, it becomes slightly more difficult. We propose to extract the whole table (except non-distributed attributes), to apply the hashing function to each tuple, and then to send the matching one (grouped into buckets) to their new site. If its cpu is free enough, hashing and sending are done by the *distribution manager*, otherwise, they are distributed among the *calculators*. Until now, new sites are determined depending on the last fragmentation and according to disk capacities [6]. As long as it remains possible, we choose a site where no fragment of the concerned relation is stored.

5. Parallel query optimization

The *parallel query optimizer* implements a one-phase randomized search strategy approach [14]. It is built up in a strict modular way, as detailed in Figure 6.

First the *transformation manager* catches up the optimal sequential execution plan from the EDBMS. It then applies a suite of local transformations to this plan, in order to find the best parallelized plan [15].

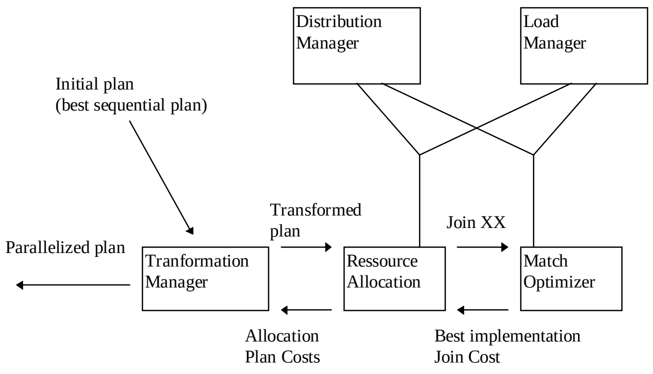

Figure 6 : The modules of the parallel query optimizer

For each operation of the plan, the *transformation manager* collaborates with the *resource allocation manager*, which optimizes the I/O, CPU and memory consumption. Therefore, it determines a first set of calculators having enough memory and CPU power to execute the operation. It then adapts this initial set to data location, in order to save I/O and communication over-costs.

At last, the *resource manager* calls the *match optimizer*, which designs the best relation access methods, i.e. the best implementation technique for the operation (e.g. hash vs nested-loop join).

All required optimization informations are extracted from the *distribution manager*, containing the actual relation partitioning, and the *load manager*, which presents the actual I/O, CPU and memory consumption on each *calculator* (see the *server* scheme in Figure 2)

The proposed architecture designs a highly extensible approach to parallel query optimization and fits particularly well in with complex queries on a decision support systems. Therefore it adapts perfectly to the requirements of the overall system, executing on the calculators complex queries, which would otherwise have overloaded the EDMS.

6. Related work

Classically, two approaches are presented in the context of parallel databases development. In the first approach, the parallel database is implemented from scratch. In the second, implementation is done by parallelising an existing sequential DBMS. University research groups are divided between these two schools. DBS3 [16] and Gamma belong to the first category, whereas Volcano [17], Midas [18] and XPRS [19] integrate parallelism within an existing DBMS.

Concerning industrial vendors, we mainly find the second approach, like IBM products [20], Oracle [21], and Informix [22]. This is not astonishing as the software can be more quickly developed when already running codes could be reused. However the amount of work needed to achieve parallel functionnalities remains important. For example, the implementation of Volcano took about five years.

Given the present context of concurrency and users' demand, the industry cannot afford such long development delays. In such a scope we integrated our parallel extension as a whole component of the existing DBMS. Interfacing is enabled by means of the available input/output functions of this DBMS. Development costs are decreased and security functionnalities, such as backup and data recovering, are easily maintained.

7. 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, oriented toward document databases and working under relaxed update constraints. As described above, developing time is drastically decreased while using our method, as we integrate as many existing elements as possible. We propose an partial data distribution strategy to adapt distribution to the stations availability and to the EDBMS level of use. The *parallel query optimizer* and the *calculators* have been developed, and are now tested and evaluated.

Anyway, many points still remain to be developed. First, we did not adress so far the server failure problems. As we use a single existing DBMS, we cannot provide the security system of a full parallel DBMS. We first propose a bandwidth and a speed extension, not a security extension, and second we can trust the robustness of the existing system (lack of software failures, security copies...). To ensure security and fastness of reaction in case of an accident, we plan to study and exploit duplication possibilities. Finally, we have to study and integrate document databases specificities into our system, by looking at the adequation of existing fragmentation techniques to our needs.

We can conclude by underlining the fact that our system, while originaly 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.J. DeWitt and J. Gray**. Parallel Database Systems : the Future of High Performance Database Systems. *Communications of the ACM*, 35(6) :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. Scheider and D.J. 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 21st VLDB Conference*, Zurich, Switzerland, 1995.

[5] **D. Chamberlin and F. Shmuck**. 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 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*, chapter 4, An Integrated Approach of Performance Prediction on Networks of Workstations. K. Bagchi, J. Walrand and G. Zobricht, Eds , Gordon and Breach, 1996.

[8] **D. Schneider et al**. Practical Skew Handling in Parallel Joins. *In Proceedings of the 18th VLDB Conference*, Vancouver, British Columbia, August 1992.

[9] **F. Douglis and J. Ousterhout**. Transparent Process Migration : Design Alternatives and the Sprite Implementation. *Software - Practice and Experience*, 21(8) :757-785, August 1991.

[10] **M. Mutka and M. Livny**. The Available Capacity of a Privately Owned Workstation Environment. *Performance Evaluation*, 12(4) :269-284, July 1991.

[11] **R.H. Arpaci et al**. *The interaction of Parallel and Sequential Workloads on a Network of Workstations*. CS-94-838, UC Berkeley, 1994.

[12] **G. Copeland and T. Keller**. Data Placement in Bubba. *In Proceedings of the ACM SIGMOD Conference*, pp 99-108, Chicago, IL, May 1988.

[13] **G. Graefe.** Query Evaluation Techniques For Large Databases. *ACM Computing Surveys, 25*(2), June 1993.

[14] **R.S.G. Lanzelotte, P. Valduriez and M. Zaït**. Industrial Strength Parallel Query Optimization : Issues and Lessons. *Information Systems - An International Journal*, 1994.

[15] **L. Brunie and H. Kosch**. Control Strategies for Complex Relationnal Query Processing in Shared Nothing Systems. *ACM SIGMOD Records,* 25(3), September 1996.

[16] **P.Valduriez, M. Couprie and B. Bergstein**. Prototyping DBS3, Shared-Memory Parallel Database System. *In Proceedings of the 1st international Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1991.

[17] **G. Graefe and D.L. Davison**. Encapsulation of Parallelism and Architecture Independance in Extensible Database Query Processing. *IEEE Transactions on Software Engineering*, 19(7), July 1993.

[18] **G. Bozas et al.** On Transforming a Sequential SQL DBMS into a Parallel One : First Results and Experiences of the MIDAS Project. *In LLNCS 1124 Springer, Eds, EUROPAR'96*, pp 881-887, August 1996.

[19] **W. Hong.** *Parallel Query Processing Using Shared Memory Multiprocessors and Disk Arrays*. PhD Thesis, University of California, Berkeley, August 1992.

[20] **C.K. Baru et al.** DB2 Parallel Edition. IBM Systems Journal, 34(2), 1995.

[21] **B. Linder**. Parallel Databases Processing in ORACLE. *In PDIS-93*, San Diego, USA, 1993.

[22] **B. Gerber**. Informix Ob Line XPS. *In Proceedings of the International Conference on Management of Data*, ACM Sigmod Record, 1995.