

A PC-NOW Based Parallel Extension for a Sequential DBMS

Matthieu Exbrayat and Lionel Brunie

Laboratoire d'Ingénierie des Systèmes d'Information
Institut National des Sciences Appliquées, Lyon, France
Matthieu.Exbrayat@lisi.insa-lyon.fr, Lionel.Brunie@insa-lyon.fr

Abstract. In this paper we study the use of networks of PCs to handle the parallel execution of relational database queries. This approach is based on a parallel extension, called *parallel relational query evaluator*, working in a coupled mode with a sequential DBMS. We present a detailed architecture of the parallel query evaluator and introduce Enkidu, the efficient Java-based prototype that has been build according to our concepts. We expose a set of measurements, conducted over Enkidu, and highlighting its performances. We finally discuss the interest and viability of the concept of parallel extension in the context of relational databases and in the wider context of high performance computing.

Keywords: Networks of workstations, Parallel DBMS, Java

1 Introduction

Parallelizing Database Management Systems (DBMS) has been a flourishing field of research for the last fifteen years. Research, experiment and development have been conducted according to three main goals. The first one is to accelerate heavy operations, such as queries involving the confrontation of huge amounts of data (by parallelizing elementary operations over the nodes and distributing data among the disks – I/O parallelism). The second one is to support a growing number of concurrent users (by dispatching connections and queries among the processors). The third goal is to offer a high level of fault tolerance, and therefore to guarantee the availability of data, for instance in the context of intensive commercial transactions (e.g. by using RAID techniques).

The very first parallel DBMSs (PDBMSs) were based on specific machines, such as Gamma [1] and the Teradata Parallel Database Machine [2]. The next logical step appeared in the middle of the 90's, with such PDBMSs as Informix On Line XPS [3], IBM DB2 Parallel Edition [4] and Oracle 7 Parallel Server [5], which were designed to work on standard (parallel) machines. Some of these systems (e.g. Informix), were defined as running on “Networks of Workstation”. Nevertheless, this definition was quite erroneous, as they were mainly designed to work on high-end architectures, such as the IBM SP2 machine. The very last developments, like Oracle 8 Parallel Server [6] take advantage of recent cluster architectures, and partially hide the management of parallelism (the administrator only has to define the list of nodes and disks to be used). It is in fact

noticeable, that the use of a network of PCs to support a PDBMS has been poorly studied. We can cite Midas [7] (parallel port of a sequential DBMS to a LAN of PCs), and the 100 Node PC Cluster [8] database (developed from scratch).

Nevertheless, while the very large majority of studies and products consist in fully porting sequential DBMSs to parallel architectures, we estimate that networks of PCs could lead to a new approach of DBMS parallelization, considering the network of PCs as a parallel extension for an existing sequential DBMS. This extension, named *coupled query evaluator*, consists of a parallel execution component (on the network of PCs), which works together with a sequential DBMS, in order to offer both high performance for query evaluation (on the parallel component) and coherency for data creation and modification (on the sequential DBMS).

In section 2, we will detail the architecture of our proposal. Its implementation will then be introduced in section 3. In section 4 we will present some measurements conducted over our prototype. In section 5 we will discuss the relevance and impact of the concept of parallel extension. Finally, in section 6 we will present some application domains of our extension.

2 Architecture

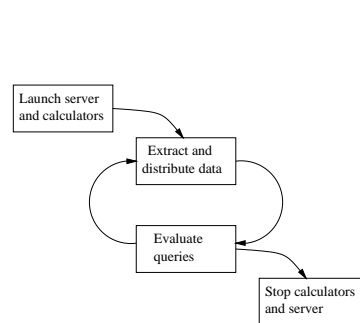


Fig. 1. Extension's basic phases

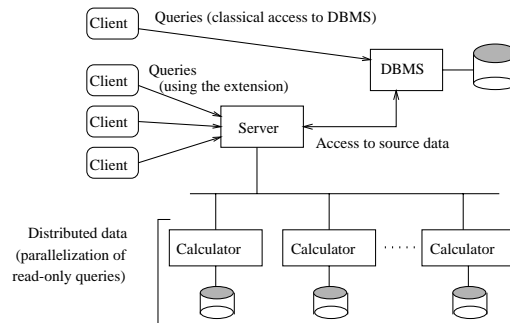


Fig. 2. General overview

2.1 General Overview

The coupled query evaluator works through two successive phases (see fig. 1). First, data is extracted from the pre-existing relational DBMS and distributed among a network of workstations. Second, this distribution is used for the parallel processing of relational queries.

The overall architecture consists of two main components (see fig. 2): the *server* and the *calculators*. The server is the access point. All tasks are submitted to and treated by it. This *server* is connected to several *calculators* which are

in charge of storing and processing redistributed data. In our architecture we assume that only one component, i.e. one calculator or the server, is running on each station (we must underline that such a choice does not bring any limitation, for instance on a SMP station, as far as a single calculator can handle several computing threads – see section 3.3).

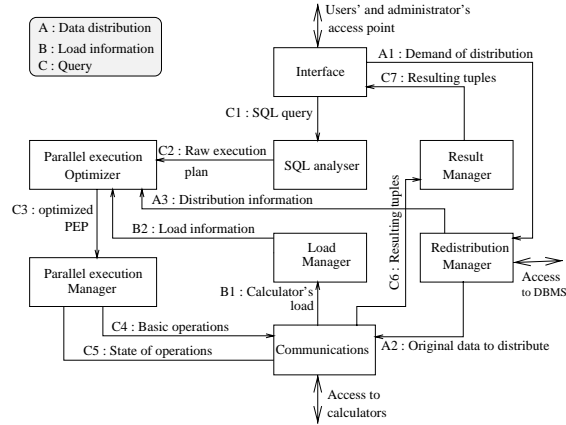


Fig. 3. Server module

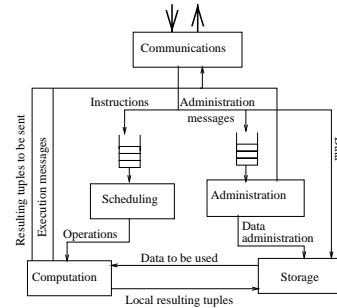


Fig. 4. Calculator module

2.2 The Server Module

The server module (see fig. 3) consists of eight components in charge of data distribution (circuit A), collection of load information (circuit B) and parallel query execution (circuit C).

Data distribution is done through the *redistribution manager* (A1), which extracts the requested data from the DBMS. Extracted data is sent to the calculators through the *communication* module (A2). Redistribution parameters are then stored by the *parallel execution optimizer* (A3).

Processor load information is regularly sent by each calculator (B1 and B2). Distribution and load information is used by the *parallel execution optimizer* in order to determine the best suited location for each operation.

Query execution is triggered by submitting a SQL query through the *interface*. This query is then translated into an internal format by the *SQL analyzer* (C1). This raw parallel execution plan (PEP) is then improved by the *parallel execution optimizer* (C2). This optimized PEP (C3) consists of basic (elementary) operators connected by flows of data and pre- and post-conditions, e.g. scheduling decisions [9]. The *parallel execution manager* analyses the PEP so that each calculator only receives the operators which take place on it (C4). The *parallel execution manager* receives (C5) processing information (e.g. end of an operator). Resulting tuples are grouped and stored by the *result manager* (C6), and then returned to the user (C7).

2.3 The Calculator Module

The calculator module consists of five components (see fig. 4). The *communication* module is similar to the one of the server module. It allows networking with the server and with all other calculators. Incoming data is transmitted to and stored by the *storage* module. Incoming instructions are transmitted to the *computation* module according to the order determined by the *scheduling* module. Intermediate results that will be used locally are transmitted to the *storage* module, while other results are sent to other calculators (intermediate results) or to the server (final results). Execution messages are also sent to the server at the end of each operator. Finally, calculators can handle administration messages (e.g. suppression of relations, shutdown of the calculator).

3 Prototyping

3.1 General Overview

Based on the architecture above, we have developed a complete prototype, named Enkidu, and written in Java, owing to the robustness and portability of this language. Enkidu is a RAM-based parallel query evaluator, which offers various distribution and execution strategies. It can be used with real data (downloaded from an existing database, or from a storage file) or with self-generated data (according to given skew parameters). Thanks to its Java implementation, Enkidu has already been used under Solaris, Linux and Windows 95.

3.2 Implementation of the Server Module

The server module mainly consists of Java code. Nevertheless, the MPO parallel execution plan optimizer [10], an external component developed in C, is currently being adapted through the Java Native Interface [11]. The server module can simulate concurrent users. This is rather important, as far as the large majority of existing academic PDBMS prototypes do not really care about concurrent queries (though DBMSs are generally supposed to support and optimize the combined load of concurrent users). Data extraction is done by the server, through the jdbc interface. Enkidu first loads the data dictionary. Then the administrator can distribute data. Extraction is done with a SQL "Select" method, due to the portability and ease of use of this method (see also section 5.1).

3.3 Implementation of the Calculator Module

The calculator module is a pure Java software. The computation module is multi-threaded: several computation threads are working on different operators. Their priority is determined according to the precedence of queries and operators. The thread with highest priority runs as long as input data remains available. If no more data is temporarily available (in a pipelined mode), secondary priority threads can start working (i.e. no time is lost waiting). Thread switching is

limited by using a coarse grain of treatment: tuples are grouped in packets, and a computation thread can not be interrupted until it finishes its current packet. Thread switching is based on a gentlemen's agreement (i.e. when a packet has been computed, the current thread lets another one start –in its priority level, or on an upper level if exists). This multi-threaded approach offers direct gains (optimized workload), and could also be useful in the context of a SMP machine, as threads could be distributed amongst nodes. With such a hardware architecture, a single calculator module could handle the whole SMP. Storage and I/O management would be managed on a single node, while other nodes would only run one (on some) computation thread(s).

We must also highlight the fact that our calculators have been designed to store data within RAM. Disks remain unused in order to avoid I/O overcosts. While this choice limits the volume of data that can be extracted and distributed, we must notice that the parallel extension is supposed to be an intermediate solution between sequential and PDBMSs. Thus, we can argue that the volume of data should remain reasonable (some GBytes at most).

3.4 Communication Issues

We chose to work at the Java sockets level, owing to their ease of use, and also because existing Java ports of MPI did not offer satisfying performance. The main problem we met did concern serialization. Serialization is a major concept of Java, which consists in automatically transforming objects into byte vectors and vice-versa. Thus, objects can be directly put on a flow (between two processes or between a process and a file). The generic serialization mechanism is powerful, as it also stores the structure of objects within the byte vector, and thus guarantees the file readability across applications. Nevertheless, this structural data is quite heavy, and introduces tremendous overcosts in the context of NOW computing. For this reason we choose to develop a specific light serialization, which only serializes data. This approach is quite similar to the one of [12], and both methods should be compared in a forthcoming paper.

4 Current Performance of the Extension Prototype

4.1 Underlying Hardware

Enkidu is currently tested over the Popc machine. Popc is an integrated network of PCs, which has been developed by *Matra Systems & Information*, as a prototype of their *Peakserver* cluster [13]. It consists of 12 Pentium Pro processors running under Linux, with 64 MByte memory each, connected by both an Ethernet and a Myrinet [14] network. The PopC machine is a computing server, which is classically used as a testbed for low- and high-level components (Myrinet optimization, thread migration, parallel programs...). In the following tests we use the Ethernet network, as it corresponds to the basic standard LAN of an average mid-size company. We are currently studying a Myrinet optimized interface. Simultaneous users are simulated by threads running concurrently on the server. To obtain reliable values, each test has been run at least ten times.

4.2 Speed-up

We realized several speed-up tests over our prototype. The one presented in this paper consists of a single hash join involving two relations (100 and 100 000 tuples). We ran these tests with 1, 5 and 10 simultaneous users. We can see on figure 5 that Enkidu offers the linear speed-ups expected with a hash join. On this figure, speed-ups seem to be “super-linear”. This comes both from the structure of the hash-join algorithm and from the fact that networking delays between the server and the calculators are included within our measurements. For this second reason, multi-user tests offer better speed-ups, as networking delays are overlapped by computation.

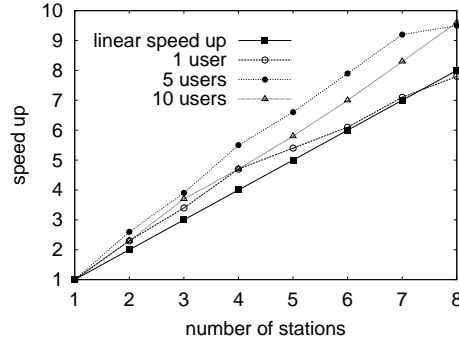


Fig. 5. Speed-up measures

4.3 Real Database Tests

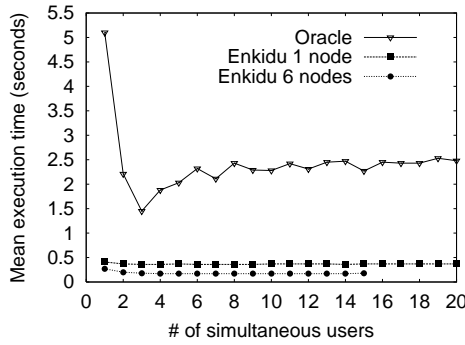


Fig. 6. Enkidu vs. oracle

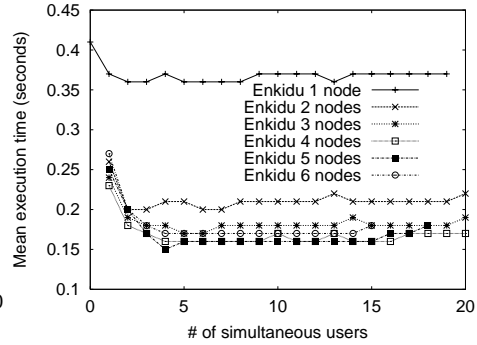


Fig. 7. Details of Enkidu execution time

Basic Database As a good speed-up could hide poor absolute performance, we also compared Enkidu to a real DBMS. The following tests are based on a

Table 1. Comparison of computation time between Oracle 7 and Enkidu

System	# of users	global exec. time (s)	mean exec. time (s)	mean exec. time * # of nodes
Oracle (1 node)	1	350	350	350
Enkidu 6 nodes	1	7.4	7.4	44.1
Enkidu 6 nodes	5	36.3	7.3	44.6
Enkidu 6 nodes	10	72.7	7.3	43.6
Enkidu 6 nodes	15	120.1	8.0	48.0
Enkidu 6 nodes	20	143.1	7.2	42.9

medicine database: the **Claude Bernard Data bank** [15]. The relatively small size of this latter (some MBytes) is counterbalanced by the lack of index (in order to simulate non pre-optimized queries). We ran our tests both on Enkidu and on our source DBMS (Oracle 7 on a Bull Estrella – PowerPC, 64 MByte RAM, AIX). The set of queries consisted in retrieving the name of medicines containing a given chemistry, for 1 to 20 concurrent users. Figures 6 and 7 highlight the good performance of Enkidu. The indicated time consists of the global response time divided by the number of users. In the context of this test, we can notice that, due to the limited size of the database, the observed speed-up is not linear (around 1.7 for 2 machines and 2.3 for 5 machines), as communication and initialization are not negligible compared to computation time.

Extended Database As the first database was quite small, we conducted a similar test with an extended database (10 times bigger for chemistries and medicines, and then 100 times bigger for the links between chemistries and medicines). As our prototype is only using RAM, we could not run this test on a single-node configuration, due to the amount of hashed data and intermediate results (performance would have suffered from the resulting swap). Thus we used a 6 nodes configuration. Concerning the Oracle platform, we only ran one user, due to both the need for very big temporary files, and the resulting swap overcosts. We can see in table 1, that using only RAM allows Enkidu (6 nodes) to compute nearly 50 times faster than Oracle. Considering the ratio computation time / number of nodes, Enkidu remains 8 times faster.

5 Discussion

5.1 Parallel Extension vs. Porting

Providing a parallel extension constitutes a specialized alternative to parallel porting. We can especially notice the differences according to the following axes:

- data storage and access: within the extension, data is loaded from a remote system (the DBMS) and stored in main memory. Within a PDBMS, data is stored on the local disks, from which it is accessed as needed;
- transaction management: the extension does not directly offer transaction management, and updates are limited.

Porting does effectively offer a complete solution, with no data load delays. Updates are automatically and relatively simply managed. Nevertheless, we see several drawbacks inherently linked to parallelization:

- development time: a complete port to parallel architectures is a heavy task, while the extension can be developed in a much faster way, due to its intentionally limited functions;
- persistence of the parallel components: a PDBMS, once initialized, uses a set of disks in a permanent manner. In the context of a network of PCs, this means that a given set of machines is dedicated to the DBMS.

As updates are managed by the DBMS, the extension must regularly update its data. As far as we mostly work with off-line applications, updates can be delayed, as long as their frequency offers a sufficient “freshness” of data (e.g. once a day). We propose to re-extract data rather than using an incremental update (which would need extra development, especially if triggers are used by the sequential DBMS – as tracking updates is then much more difficult). As an example, extracting and distributing the extended database of section 4.3 is done in less than 15 minutes: about 10 minutes for extracting data (from Oracle to the server) and 1.5 minutes for distributing it (from the server to the calculators). Technically speaking, we use a temporary file in order to handle these two phases independantly. Thus, calculators are only locked during distribution.

5.2 Toward a Generalization of the Parallel Extension Concept

The parallel extension concept could fit in a wider perspective of high performance processing. In effect, many similarities exist between our extension and recent developments, for instance in the field of scientific computing, to *extend* sequential applications by parallelizing *some* of their algorithms. Various applications, such as numerical simulation and imaging, could use sequential components during data input and light operations, and could benefit from parallel components during heavy computations. Concerning numerical simulation, parallel computing can be used for heavy computation, such as crash simulation, or fluid mechanics, while designing the structure is usually done in a sequential way. Considering image computing, the input and annotation of pictures should be done sequentially, while image fusion or filtering can benefit from parallel algorithms. In a more general way, a parallel extension could be used whenever a software alternatively executes light and heavy treatments.

6 Context

As our extension appears to be mainly interesting in the context of “read-mostly” applications, we will now give three examples of such applications: decision support systems, data mining and multimedia databases.

Within decision support systems (DSS), data are frequently manipulated off-line(i.e. data generation and manipulation are two distinct and independant

tasks). Thus, an extension can be used. As an example we will cite the well known TPC-R and TPC-H benchmarks. TPC-R [16] is a business reporting benchmark, adapted to repetitive queries, concerning a large amount (1 TByte) of data, and thus oriented toward the (static) optimization of data structures and queries. TPC-H [17] works on ad-hoc queries, i.e. various and theoretically non-predictable queries. It can thus be used with various dataset size (from 1 GByte to 1 TByte). Our system can *a priori* be used in both cases, and at least with TPC-H, as small amounts of data can be manipulated. Although our current implementation is not adapted to large datasets (data is stored in RAM), it could anyway work on databases ranging from 1 to 10 GByte, by using a cluster of PCs handling enough memory (e.g. 6 to 10 PCs, each having 256 MByte memory, could easily handle a 1 GByte database). Of course, we could also implement some existing algorithms using disks to store temporary data, such as the well known hybrid hash-join algorithm [18].

Concerning data mining, our extension could at least be used during pre-processing phases, in order to provide a fast and repetitive access to source data. It could even be used during processing, as far as [19] showed that knowledge extraction could also be done through a classical DBMS using SQL.

Concerning multimedia databases, both academic researchers [20] and industrial software developers [21, 22] are deeply implicated in delivering multimedia DBMSs. The read-mostly nature of such databases is trivial. For instance, the Medical Knowledge Bank project [23], and especially its initial and continuing medical education section, mainly involves read-only accesses.

7 Summary

In this paper we proposed and discussed the use of a parallel extension in the context of Database Management Systems and we presented the prototype we built according to our proposal. Through our tests it appeared, that this extension is a valuable alternative to the classical parallel porting of DBMSs, especially in the context of read-mostly applications.

Future work should follow two main goals: getting even better performance and developing specifically adapted algorithms. From the performance point of view, we plan to develop high performance (Myrinet-based) Java components. We also wish to upgrade our packet-based techniques toward a real macro-pipelining approach. Finally, we are trying to get faster extraction and distribution algorithms. From the applications point of view, we are currently studying some multimedia and information retrieval algorithms working over our architecture. Another important direction of research, from our point of view, consists in testing the concept of parallel extension in various fields, and to propose a global and generic definition for it.

References

1. D. Dewit, S. Ghandeharizadeh, D. Schneider, *et al.*, "The Gamma Database Machine Project," *IEEE TKDE*, vol. 2, pp. 44-62, Mar. 1990.

2. J. Page, "A Study of a Parallel Database Machine and its Performance the NCR/Teradata DBC/1012," in *Proceedings of the 10th BNCOD Conference*, (Aberdeen, Scotland), pp. 115–137, July 1992.
3. B. Gerber, "Informix On Line XPS," in *Proceedings of ACM SIGMOD '95*, vol. 24 of *SIGMOD Records*, (San Jose, Ca, USA), p. 463, May 1995.
4. C. Baru, G. Fecteau, A. Goya, *et al.*, "DB2 Parallel Edition," *IBM Systems Journal*, vol. 34, no. 2, pp. 292–322, 1995.
5. R. Bamford, D. Butler, B. Klots, *et al.*, "Architecture of Oracle Parallel Server," in *Proceedings of VLDB '98*, (New York City, NY, USA), pp. 669–670, Aug. 1998.
6. Oracle, "Oracle Parallel Server: Solutions for Mission Critical Computing," tech. rep., Oracle Corp., Redwood Shores, CA, Feb. 1999.
7. G. Bozas, M. Jaedicke, A. Listl, *et al.*, "On transforming a sequential sql-dbms into a parallel one : First results and experiences of the MIDAS project," in *EuroPar'96*, (Lyon), pp. 881–886, Aug. 1996.
8. T. Tamura, M. Oguchi, and M. Kitsuregawa, "Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining," in *SC'97*, 1997.
9. L. Brunie and H. Kosch, "Optimizing complex decision support queries for parallel execution," in *PDPTA '97*, (Las Vegas, AZ, USA), July 1997.
10. L. Brunie and H. Kosch, "ModParOpt : a modular query optimizer for multi-query parallel databases," in *ADBIS'97*, (St Petersburg, RU), 1997.
11. S. Liang, *The Java Native Interface: Programmer's Guide and Specification*. Java Series, Addison Wesley, June 1999.
12. M. Philippsen and B. Haumacher, "More Efficient Object Serialization," in *International Workshop on Java for Parallel and Distributed Computing*, (San Juan, Porto Rico, USA), Apr. 1999.
13. MatraSI, "Peakserver, the Information Server." [On-Line], Available on Internet : <http://www.matra-msi.com/ang/savoir_serv_d.htm>, 1999.
14. N. Boden, D. Cohen, R. Felderman, *et al.*, "Myrinet - a gigabit-per-second local-area network," *IEEE-Micro*, vol. 15, pp. 29–36, 1995.
15. A. Flory, C. Paultre, and C. Veilleraud, "A relational databank to aid in the dispensing of medicines," in *MEDINFO '83*, (Amsterdam), pp. 152–155, 1983.
16. TPC, *TPC Benchmark R (Decision Support) Standard Specification*. San Jose, CA: Transaction Processing Performance Council, Feb. 1999.
17. TPC, *TPC Benchmark H (Decision Support) Standard Specification*. San Jose, CA: Transaction Processing Performance Council, June 1999.
18. D. Schneider and D. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," in *Proceedings of ACM SIGMOD '89*, (Portland, Oregon, USA), pp. 110–121, June 1989.
19. I. Pramudiono, T. Shintani, T. Tamura, *et al.*, "Mining Generalized Association Rule Using Parallel RDB Engine on PC Cluster," in *DaWak'99*, (Florence, Italy), pp. 281–292, Sept. 1999.
20. H. Ishikawa, K. Kubota, Y. Noguchi, *et al.*, "Document Warehousing Based on a Multimedia Database System," in *ICDE'99*, (Sydney, Australia), pp. 168–173, Mar. 1999.
21. Oracle, "Oracle Intermedia: Managing Multimedia Content," tech. rep., Oracle Corp., Redwood Shores, CA, Feb. 1999.
22. Informix, "Informix Media 360," tech. rep., Informix, Menlo Park, CA, Aug. 1999.
23. W. Sterling, "The Medical Knowledge Bank: A Multimedia Database Application," *NCR Technical Journal*, Aug. 1993.