# ParGRES: a middleware for executing OLAP queries in parallel

Marta Mattoso[1], Geraldo Zimbrão[1,3], Alexandre A. B. Lima[1], Fernanda Baião[1,2], Vanessa P. Braganholo[1], Albino Aveleda[1], Bernardo Miranda[1], Bruno Kinder Almentero[1], Marcelo Nunes Costa[3]

.

[1] COPPE, Federal University of Rio de Janeiro, Brazil
[2] Department of Applied Informatics, UNIRIO, Brazil
[3] DCC/IM, Federal University of Rio de Janeiro, Brazil

pargres@nacad.ufrj.br

http://forge.objectweb.org/projects/pargres

**Technical Report ES-690**
**December, 2005**

## ABSTRACT

ParGRES is a middleware aimed to efficiently process heavy weight queries, typical of OLAP, on top of a database cluster. ParGRES achieves query processing speed-up through intra- and inter-query parallelism in a PC cluster environment with database replication and virtual partitioning. It accelerates both individual queries and system throughput. Our experimental results show that ParGRES yields super-linear or near-linear speed-up. ParGRES middleware keeps application and database autonomy. As a result, it offers a non-intrusive migration solution from sequential to a parallel environment. Currently, ParGRES uses PostgreSQL, but it is not DBMS dependent, and has a Web administration tool. The main features of ParGRES are: automatic parsing of SQL queries to allow for intra-query parallel execution; query processing with inter- and intra-query parallelism; virtual dynamic partition definition; result composition; update processing; and dynamic load balancing. The main contribution of ParGRES is to combine inter and intra-query parallelism with dynamic load balancing for virtual partitions, all within an open source cost-effective solution.

## 1. INTRODUCTION

The performance of information systems is becoming critical in organizations, due to the increasing volume of data that must be analyzed by applications, and to the complexity of the queries they submit to the database system (DBMS). The performance in accessing stored data has great influence in OLAP (On-Line Analytical Processing) applications, which access huge datasets through heavy-wight queries [3]. In this kind of application, the database is periodically (rather than online) updated [14]. Database tuning is very difficult, since queries are *ad-hoc*.

Parallel processing has been successfully used to improve performance of heavy-weight queries, typically by replacing the software and hardware platforms with higher computational capacity components (e.g. parallel servers and/or parallel DBMSs). This approach requires adapting the database design and applications from the sequential to the parallel environment. Migrating applications is complex (sometimes impossible), since it may require modifications to the source code. In addition, often it requires the expansion of the computational environment and the application modification, which can be very costly. A cheaper alternative is the use of PC clusters. However, the costs can still be high because some solutions require specific software (DBMS) or hardware (e.g. SAN – Storage Area Network).

Another existing approach is called “database cluster” [11]. This approach uses a sequential DBMS at each cluster node. These DBMS are used as *black boxes*, that is, they are ordinary sequential DBMS with no modification in their source code, and no additional functionality to improve their use in clusters. It is all based on off-the-shelf hardware and software components. In addition, this approach requires no changes in the database design or in the application source code, thus greatly reducing the migration cost of applications and databases to the parallel environment.

ParGRES [6] is a database cluster acting as a middleware between the application and the DBMS. It is capable of coordinating the access to data to obtain the desired parallelism using a standard PC cluster. ParGRES parallelism is devoted to SQL queries that take a long time to be processed, e.g, typical OLAP queries. We propose a unique database cluster solution that: combines inter- and intra-query parallel processing techniques; deals with updates; and performs load balancing between the system nodes. Our experimental results show that ParGRES yields super-linear or near-linear speed-up. In this implementation, we use the PostgreSQL [10] DBMS at each cluster node. However, our parallel strategy is based on SQL only and it does not use any specific feature of PostgreSQL. Thus, organizations have more flexibility in choosing or keeping their own DBMS platform.

This work is organized as follows. Section 2 describes ParGRES architecture. Section 3 details the computational environment where ParGRES was developed, while Section 4 describes the aspects we will cover during the demonstration. Section 4 shows our experimental results and section 6 analyzes related work. Finally, we conclude in Section 7.

## 2. PARGRES ARCHITECTURE

As in other database cluster solutions, ParGRES is a middleware that orchestrates the parallel execution of queries using DBMS instances at the cluster nodes. ParGRES, has a distributed architecture (Figure 1): its components are distributed over the cluster thus avoiding the overload of the coordinator node. This architecture is based on the algorithms proposed by Lima [5].

There are global and local components. Global components execute tasks that involve several cluster nodes, while local ones execute tasks in a single node. The global components are the *Mediator* and the *Cluster Query Processor* (CQP). The local components are the *Node Query Processor* (NQP) and the DBMS.

The most important component is the CQP, which acts as the coordinator of all remaining components. Since most PC clusters have a single node that is accessible to external applications (the entry node), the Mediator component is allocated in this node (it must communicate with the applications). The Mediator acts just like a proxy, receiving requests from the applications, passing them to the CQP and passing back CQP responses to the applications. The allocation of the Mediator in the entry node gives us total flexibility when physically allocating the CQP, which improves the overall environment availability. If the node where the CQP is allocated fails, it can simply be re-instantiated in another node. NQP locally coordinates query execution at the DBMS and helps CQP during load balancing.

ParGRES executes four types of tasks: (i) SQL query parsing, to enable parallel execution, (ii) query processing with inter/intra-query parallelism, (iii) result composition and (iv) update processing, detailed as follows.

**SQL query parsing.** CQP has a *Translator* that contains a syntactic analyzer to parse SQL commands from the client application. It uses a context-free grammar for SQL-99. Commands not parsed by this grammar are sent directly to the DBMS. The information generated by the Translator includes: (i) a set of relations and attribute names referenced by the query that may be used in the intra-query parallelism; (ii) information needed by the CQP to perform result composition; (iii) a set of attributes used in aggregation operations. Once the Translator identifies relations and attribute names, CQP decides which parallel strategy to use.

**Query processing with inter/intra-query parallelism.** CQP is responsible for analyzing the queries and deciding the type of parallelism and the subset of nodes that will be used for processing each query. To do so, it uses information from the *Catalog*. This Catalog is very simple and only stores information needed to implement the adaptive virtual partitioning as a non intrusive technique. The catalog does not need specific information of the DBMS, and maintains the philosophy of using the DBMS as a “black-box” component.

The intra-query (**intra-q**) parallel strategy decomposes complex queries into sub-queries that will be executed in parallel. Each sub-query runs over a different data fragment. Since each sub-query is sent to a different node, it can be executed in parallel. In the inter-query (**inter-q**) parallel strategy, distinct queries are executed concurrently in the DB cluster, one at each cluster node.

Inter-q parallelism implementation is almost straightforward. CQP sends the query to the NQP of the node with the smallest number of pending tasks. NQP then passes the query to the DBMS. The result follows the inverse path to the client application. In the case of an update operation, CQP blocks the execution of queries, and sends the update to the NQP of each node, to guarantee consistency. After all NQPs confirm the execution of the update, CQP allows the execution of new queries.

To provide high performance in heavy-weight queries, ParGRES implements intra-q parallelism using the adaptive virtual partitioning (AVP), proposed by Lima et al. [4], with full database replication. AVP depends on the existence of a clustered index over the relations involved in the query. This kind of information is in the Catalog. It stores the names and cardinalities of the relations that have clustered indexes, attributes which have a clustered index, and the range of values of each of such attributes. Since each relation can have a single clustered index, the amount of information stored in the Catalog is not large.

CQP decides which attribute will be used in the virtual partition of the intra-q strategy. Then the original query is re-written in sub-queries by the Translator. Those sub-queries are a version of the original query containing a predicate that determines the ranges of the virtual partitioning. Then, CQP chooses an NQP to globally coordinate the execution of the query. The chosen NQP receives the sub-queries from CQP and interacts with the other NQPs, called *participating* NQPs, to execute them. Each participating NQP receives its sub-queries and adaptively fine tune the virtual partitions locally. Each new query processed with the intra-q strategy may have a different NQP as a coordinator thus allowing the CQP to globally balance the load over the cluster.

Due to the value distribution of attributes used by a query, the initial workload of all nodes may be non-uniform. In addition, results of intermediary operators of a query may also lead to skew. Our non-intrusive approach that treats the DBMS as a black-box component makes it difficult to prevent skew. Thus, the ParGRES dynamic load balancing addresses skew during intra-q execution. The NQPs help this balancing by exchanging messages among themselves to redefine virtual partitions. It is implemented using a distributed technique proposed by Lima [5]. The results presented in [5] show that this is a very efficient technique, especially for cases of extreme skew.

In the intra-q parallelism, the coordinator NQP allocates the participating NQPs and sends a local query execution plan to each of them. Each participating NQP process its plan and generates a partial result, which is sent to the coordinator. After receiving the partial results from all participating NQPs, the coordinator finishes the result composition and sends it to CQP that forwards it to the client application.

Since several queries can be processed at the same time in the cluster (and in the same node), the inter-q parallelism is used with the intra-q parallelism. One of the advantages of this combination is that some queries may be of low-cost, which is not adequate for intra-q parallelism. In this case, inter-q parallelism is chosen to improve the system throughput.

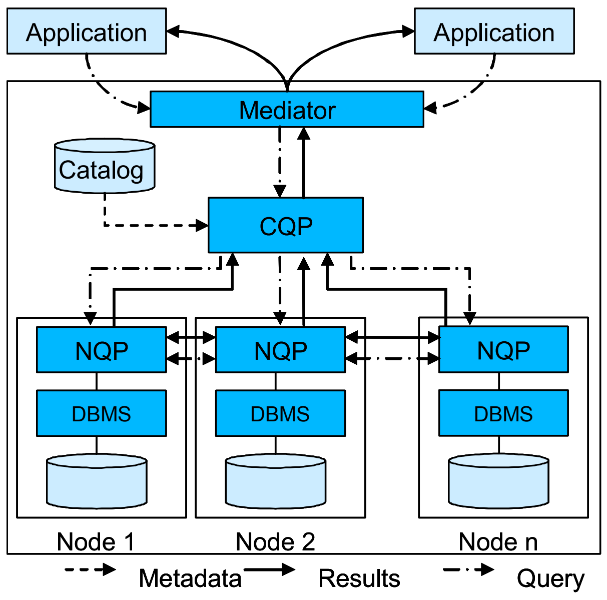


**Figure 1: ParGRES architecture**

**Result composition.** ParGRES does result composition by adapting the two-phase aggregation algorithm proposed by Shatdal and Naughton [12]. Our algorithm uses parallel processing in this composition, minimizing the communication between nodes. In the first phase, the nodes aggregate the groups returned by the local sub-queries. In the second phase, the groups are distributed to their respective nodes through a hash function. Finally, each node sends its subset of the global result to the coordinator node, which executes their union. When the query involves the “order by” clause, an additional ordering phase is needed, which is also executed in parallel at the cluster nodes.

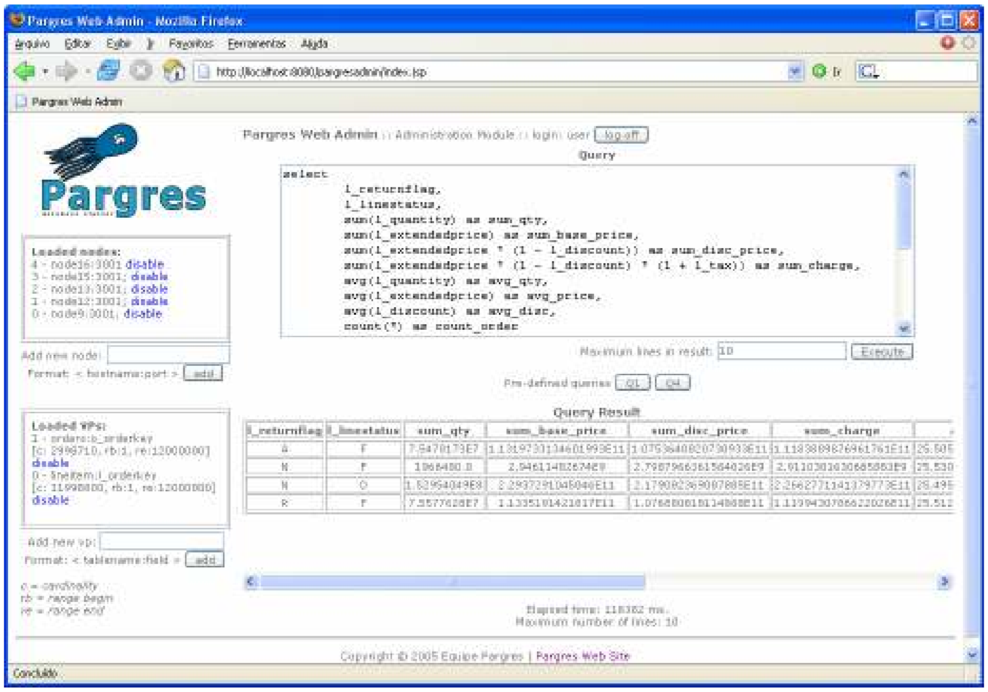
**Update processing.** Although ParGRES mainly focuses on read-only query processing, typical of OLAP, updates may also be sent by the client application. The parallel execution of updates and queries with the intra-q parallelism in an environment with total data replication has to be carefully conducted to avoid inconsistent results. Since updates in OLAP environments are usually are fast and executed at pre-determined times, ParGRES adopts a strong consistency policy: it does not allow the concurrent execution of updates and queries. To implement this policy, ParGRES has a scheduler that orders queries and updates. While updates are processed, all the remaining queries coming from the application are blocked. When there are just read-only queries, CQP allows them to execute in parallel.

## 3. COMPUTATIONAL ENVIRONMENT

The use of ParGRES assumes a computational environment of three layers: the application layer (an OLAP tool), the ParGRES layer, and the database layer (with DBMSs installed in the nodes of a PC cluster. Each DBMS accesses its local database, in which the data cubes have already been generated and are ready to be queried). We assume the OLAP tool acquires data with SQL.

The communication of ParGRES and the OLAP tool is simple. It is only necessary to redirect the configuration of the database server accessed by the OLAP tool to the cluster in which ParGRES is being executed. OLAP queries are intercepted by taking advantage of a JDBC driver in a transparent way. The Translator was written using the BYacc/J tool, the traditional Yacc adapted to generate Java code.

ParGRES was developed in Java, and tests were executed in a PC cluster following a shared-nothing architecture. ParGRES does not need any specific cluster hardware such as SAN or high-speed network adapters. In the current implementation, ParGRES is using version 8.0 of PostGRESQL as the DBMS at each cluster node. The communication between ParGRES and each DBMS is done through JDBC. Besides, the communication between each internal module of ParGRES is done through RMI.


Pargres Web Admin - Mozilla Firefox
Arquivo Editar Exibir Ir Favoritos Ferramentas Ajuda
http://localhost:8080/pargresadmin/index.jsp
Pargres Web Admin
Pargres
Pargres Web Admin :: Administration Module :: login: user
log off
Query
Maximum lines in result:
Execute
Pre-defined queries
Q1
Q4
Add new node:
Format: < hostname:port >
add
Add new vp:
Format: < tablename:field >
add
Concluído

Loaded nodes:
4 - node16:3001 disable
3 - node15:3001; disable
2 - node13:3001; disable
1 - node12:3001; disable
0 - node9:3001; disable

Loaded VPs:
1 - orders:o_orderkey
[c: 2999710, rb:1, re:12000000]
disable
0 - lineitem:l_orderkey
[c: 11998800, rb:1, re:12000000]
disable

c = cardinality
rb = range begin
re = range end

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
```

Maximum lines in result: 10

Query Result

| l_returnflag | l_linestatus | sum_qty | sum_base_price | sum_disc_price | sum_charge | |
|---|---|---|---|---|---|---|
| A | F | 7.5470173E7 | 1.1319733134601993E11 | 1.0753640820730933E11 | 1.1183809876961761E11 | 25.505 |
| N | F | 1968480.0 | 2.946114826747E9 | 2.7987966361564026E9 | 2.9110303630685063E9 | 25.530 |
| N | O | 1.52954049E8 | 2.2937291045046E11 | 2.1790823690078E11 | 2.2662771141379773E11 | 25.495 |
| R | F | 7.5577628E7 | 1.1335101421017E11 | 1.0768808181148E11 | 1.1199430786622026E11 | 25.512 |

Elapsed time: 118362 ms.
Maximum number of lines: 10

Copyright © 2005 Equipe Pargres | Pargres Web Site

Figure 2: A screenshot of Pargres administration tool.

## 4. DEMONSTRATION

A web-based administration tool was developed in Java/JSP for ParGRES (Figure 2). It allows for the user to remotely interact with the middleware to perform on-line query submissions and administration tasks. It is possible, for example, to dynamically add and remove NQPs to/from a running ParGRES instance. If new tables are available for virtual partitioning, it is possible to dynamically update ParGRES Catalog to use intra-q parallelism to process queries based on such tables from that moment on.

During our demonstration session, we intend to use the administration tool for showing all these features. We will dynamically change ParGRES configuration in many ways. Also, we will demonstrate its query processing capabilities by showing how ParGRES parallel techniques effectively improve query processing performance. We will compare the performance of parallel and sequential executions of the same query.

## 5. EXPERIMENTAL EVALUATION

In this section, we evaluate ParGRES performance while processing OLAP queries in different cluster configurations. Our tests were based on the TPC-H benchmark [14], specific for *ad-hoc* OLAP applications. We generated the database according to TPC-H specifications using a scale factor of 5, which gave us a database of approximately 11 GB (including all indexes). Clustered indexes based on the first attribute of the primary key were generated for each fact table (Orders and LineItem). They are necessary for the adaptive virtual partitioning, implemented by ParGRES. Indexes were also generated for all other primary and foreign keys. No other indexes were created, as determined by TPC-H.

We chose a subset that we think it is quite representative for OLAP applications: queries Q1, Q3, Q4, Q5, Q6, Q7, Q8, Q12, Q14 and Q19. Those are the queries that can be processed with the intra-q strategy. Q1 has a very low selective predicate that is satisfied by almost all tuples of the largest fact table (LineItem). Besides, it performs a lot of aggregation operations. Q3 joins the two fact tables and a dimension. It generates a high-cardinality result. Although TPC-H specifies only the first 10 tuples must be returned, we used the entire result in our tests in order to stress ParGRES. Q4 is based on the Orders fact table and performs a sub-query on the LineItem table. Q5 joins both fact tables and 4 dimensions. As Q1, Q6 is based only on LineItem table. However, it has a high-selective predicate and only one aggregation function, without a "group by" clause. Q7 and Q8 have a sub-query in the "from" clause. In both cases, the two fact tables are used but Q8 use more dimensions than Q7. Q12 performs a join between the fact tables only. Q14 performs a join between the LineItem table and a relatively large dimension. Q19 is similar to Q14 but presents disjunctive clauses in its predicate.

Our tests were performed on top of a 32-node shared-nothing cluster gently provided by the Paris Team at INRIA [9]. Each node is configured with two 2.2 GHz Opteron processors, 2 GB RAM and 30 GB HD. Nodes are interconnected through a Gigabit Ethernet network. During our experiments, each node ran an instance of the PostgreSQL 8.

We show here the results obtained during our speedup experiments. Our goal is to evaluate the speedup achieved by ParGRES while processing isolated queries with different number of nodes (from 1 to 32). Each query was run ten times for each cluster configuration. Then, we took the average time of the last nine runs to make our analysis (the first one was not considered). Results are shown on Figure 3. Execution times in Figure 3 are normalized. Each value was produced by taking the average execution time obtained for the query and dividing it by the largest average execution time obtained for the same query. Figure 3 also shows the reference linear speedup.

The results present excellent speedup. Only Q5 presents sub-linear performance. Q5 performs a join between many dimensions and the fact tables, which makes it both CPU and IO bound. Unlike the other queries, Q5 tuple access is very intensive and random and does not take advantage on the increase of memory space through additional processors. For all other queries from 8 nodes on, the virtual partitions of both fact tables accessed by each node start to fit in main memory, reducing the number of disk accesses. This explains the good speedup obtained.

With 32 nodes, execution times obtained for Q1 and Q3 are respectively only 0.78% and 0.13% higher than the times that would be obtained if linear speedup was achieved. The worst case was for Q5, for which a sub-linear speedup was obtained, for the same reasons explained above. Even in such case, the execution time was only 1.95% above the time expected with linear speedup. For all other queries, super-linear speedup was obtained.

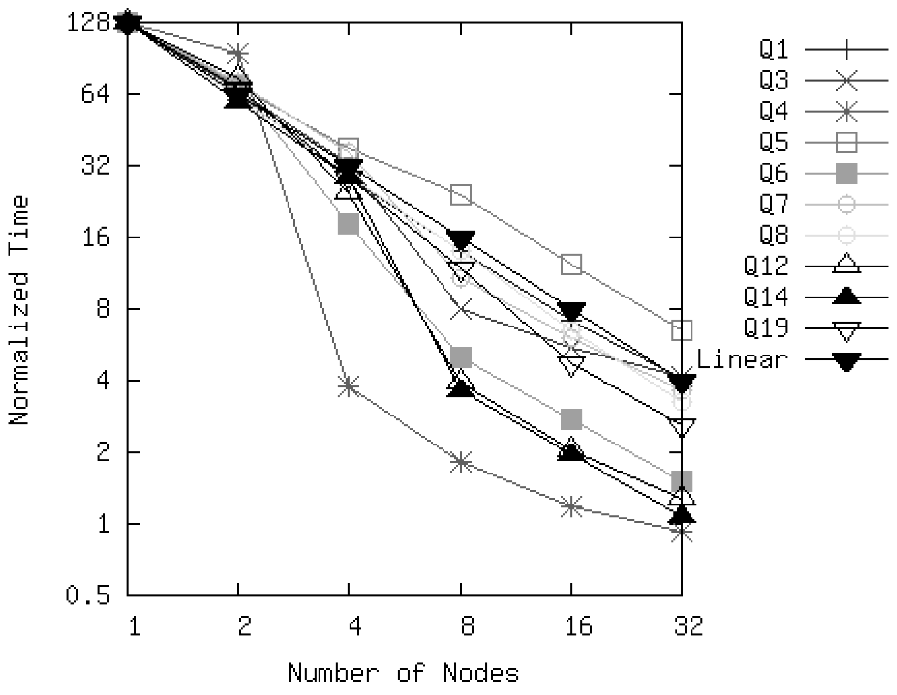


**Figure 3: Speedup experiments – query execution times**

## 6. RELATED WORK

There are several solutions to improve performance in database query processing through intra-q or inter-q parallelism, but not both. Inter-q parallelism is the focus of C-JDBC [1] and MySQL Cluster [8]. Solutions for intra-q parallelism are proposed by Oracle, DB2 and Teradata [13], but are all proprietary and DBMS and/or hardware specific.

C-JDBC [1] is an open-source database cluster based on inter-q parallelism only. This kind of parallelism accelerates the processing of several small concurrent transactions, which typically appear in OLTP environments. Thus it improves DBMS throughput. However, heavy-weight queries, typical of OLAP environments, take a long time to process, usually minutes or hours. For this kind of query, very well represented by the TPC-H benchmark [14], inter-q parallelism cannot reduce the individual query processing time and intra-q parallelism is the adequate solution.

MySQL Cluster [8] is a Free Software solution that supports only inter-query parallelism. Furthermore, it presents a strong limitation: the database is required to fit in the available amount of memory of the cluster (the sum of the main memory available in each cluster node). This is because MySQL Cluster explores in-memory processing and log writes to accelerate query processing.

Teradata [13] is a proprietary DBMS that employs massive parallelism for query processing. It provides for both intra- and inter-query parallelism and there are results showing excellent performance when processing queries on huge-sized databases. The main drawback is, besides being proprietary software, Teradata also requires expensive proprietary hardware to run.

PowerDB [11] was one of the first projects to present the concept of a "database cluster" and to propose algorithms to process queries in parallel. However, the project was discontinued and it is proprietary. Besides, the intra-q algorithm of PowerDB is not robust, since the speed-up in intra-q processing depends on the DBMS being used. In ParGRES, we have extended and improved PowerDB's techniques following the open-source ideas. In particular, we have improved the intra-q data partition of PowerDB, and added dynamic load balancing.

ParGRES innovates by combining both intra-q and inter-q parallelism, based on the parallelization algorithms proposed by Lima [5]. Additionally, ParGRES includes some interesting features to these algorithms: (i) automatic SQL query translation;

(ii) support of update operations; (iii) open-source components, among others. These features facilitate the migration of sequential applications to our parallel solution in PC clusters.

In the Free Software field, Mondrian [7] is a system focused on OLAP analysis. Mondrian can benefit from ParGRES parallelism by including the middleware between the OLAP application and the DBMS that processes its queries. As so, it is complimentary to our approach. Still in the OLAP processing area, the Panda project [2], among others, addresses the parallel generation of the data cubes in parallel environments. ParGRES is not specific to OLAP queries. Besides, our tool aims at accelerating the performance of in production applications. Thus, the database and the cube constructions are out of the scope of our work.

## 7. REMARKS AND FUTURE WORK

ParGRES is an open-source software that provides parallelism to efficiently execute heavy-weight queries over database clusters. ParGRES implements intra-q and inter-q parallelism and uses the adaptive virtual partitioning with full database replication. ParGRES is a low cost solution to the performance problem of applications that access large volumes of data through complex queries, because it uses black-box DBMS, it avoids the need to redo the physical design of the database when migrating to the parallel environment, and it does not require any specific hardware. The communication between ParGRES and the DBMS uses the SQL standard, so any SQL DBMS implementation can be adopted.

The need for full replication of the database may be pointed as a weak point of our approach. However, it is important to notice that the cost of non-volatile storage has been dropping drastically in the past years. In addition, maintaining consistency of applications with controlled updates does not invalidate full replication. This makes our approach economically attractive even for large databases. Besides, full replication increases the system availability and fault tolerance.

There are other solutions based on database clusters. However, the differential of ParGRES is that it provides efficient intra-q parallelism in a transparent way, thus reducing the response time of each high-cost query sent by the application.

## 8. REFERENCES

[1] Cecchet, E., Marguerite, J., and Zwaenepoel, W. (2004), “C-JDBC: Flexible Database Clustering Middleware”, In: Freenix 2004: USENIX Annual Technical Conference, Boston, USA, pp. 9-18.

[2] Chen, Y., Dehne, F., Eavis, T., Rau-Chaplin, A. (2004), "Parallel ROLAP Datacube Construction on Shared Nothing Multi-Processors", Journal of Parallel and Distributed Databases, 15 (3), pp. 219-236.

[3] Gorla, N. (2003), “Features to Consider in a Data Warehousing System”, Comm ACM,46(11),pp. 111-115.

[4] Lima, A. A. B., Mattoso, M. and Valduriez, P. (2004), “Adaptive Virtual Partitioning for OLAP Query Processing in a Database Cluster”, In: Proc 19th SBBD, Brasília, Brazil, pp. 92-105.

[5] Lima, A. A. B. (2004), “Intra-Query parallelism in database clusters”, DSc Thesis, COPPE/UFRJ, Brazil (in portuguese).

[6] Mattoso, M., Zimbrao, G., Lima, A. A. B., Baião, F., Braganholo, V., Aveleda, A., Miranda, B., Almentero, B., Costa, M. (2005), “The ParGRES Project”, url: http://forge.objectweb.org/projects/pargres.

[7] Mondrian (2005), “Mondrian OLAP Server”, url: http://mondrian.sourceforge.net/.

[8] MySQL Cluster (2005), url: www.mysql.com/products/cluster

[9] Paris Project., url: www.irisa.fr/paris/General/cluster.htm.

[10]PostgreSQL v.8.0. (2005), url: http://www.postgresql.org/download/.

[11]Röhm, U., Böhm, K., Schek, H.-J., et al. (2002), FAS - A Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components, VLDB, Hong Kong, pp. 754-765.

[12]Shatdal, A., Naughton, J. (1995), “Adaptive Parallel Aggregation Algorithms”, SIGMOD, pp.104-114.

[13]Teradata (2005), url: www.teradata.com.

[14]TPC (2003), “TPC BenchmarkTM H – Revision 2.1.0”, url: www.tpc.org.