# High-performance Query Processing of a Real-world OLAP Database with ParGRES ·

Melissa Paes[1, 2], Alexandre A.B. Lima[3], Patrick Valduriez[4], Marta Mattoso[1]

[1]COPPE, Federal University of Rio de Janeiro, Brazil
[2]Brazilian Institute of Geography and Statistics, Brazil
[3]UNIGRANRIO, Brazil
[4]INRIA and LINA, University of Nantes, France
melissa@cos.ufrj.br, abento@unigranrio.com.br, Patrick.Valduriez@inria.fr, marta@cos.ufrj.br

**Abstract.** Typical OLAP queries take a long time to be processed so speeding up the execution of each single query is imperative to decision making. ParGRES is an open-source database cluster middleware for high performance OLAP query processing. By exploiting intra-query parallelism on PC clusters, ParGRES has shown excellent performance using the TPC-H benchmark. In this paper, we evaluate ParGRES on a real-world OLAP database. Through adaptive virtual partitioning of the database, ParGRES yields linear and very often super-linear speedup for frequent queries. This shows that ParGRES is a very cost-effective solution for OLAP query processing in real settings

## 1. Introduction

A Data Warehouse (DW) is a large database repository that integrates data from different sources and provides a single view of all or part of the business data collection in order to improve data analysis. Online analytical processing (OLAP) is a multidimensional approach to analyze data and is used much in DW. This analysis is done through complex queries with high processing costs. The processing of a sequence of such queries can take hours or days. Often, this time-frame is above the deadline the decision makers have to obtain the required analysis.

One of the challenges for OLAP applications is the reduction of individual query processing time. To improve performance in these applications, high-performance parallel database systems can be used [22]. Parallel database systems exploit inter- and intra-query parallelism. Inter-query parallelism runs each query sequentially, but many queries in parallel, thus improving overall throughput. Inter-query aims at OLTP queries running as many concurrent queries in parallel as possible. However, in OLAP, intra-query parallel

· Candidate to the Best Student Paper Award.

processing is mandatory. There is an explicit order and sequential dependency between related queries. Running these queries in parallel will not lower the long running time of each query. The problem of using parallel database systems for high performance query processing is the vendor dependency and costs involved in software, hardware, and physical database design.

The database cluster approach [1, 3, 8, 9] improves the performance of queries by applying parallel processing on top of a cluster of databases. High-performance comes with a low cost implementation on top of PC clusters. Akal *et al.* [1] define a database cluster (DBC) as a standard computer cluster (a cluster of PC nodes) running a Database Management System (DBMS) instance at each node. A DBC middleware intercepts queries from an application and manages query execution by taking advantage of the DBC.

Following the DBC approach, ParGRES [9, 12] has been developed as a middleware to provide for intra-query parallel processing on DBC. ParGRES is an open-source middleware between the application and DBMS that intercepts SQL queries to exploit intra-query parallelism. It orchestrates its execution with the help of the sequential DBMS, each running at a cluster node. ParGRES can also provide for inter-query parallelism. To preserve the sequential DBMS execution, virtual partitioning on a replicated database has been proposed in [1] to achieve intra-query parallelism. ParGRES builds upon the virtual partitioning technique by creating an adaptive virtual partitioning that also allows for load-balancing. These features make ParGRES a unique solution for OLAP queries among several DBC systems, like PowerDB [14], C-JDBC [3], and Sequoia [18],

The efficiency of ParGRES has been validated through experiments with typical queries of the TPC-H benchmark [20]. TPC-H represents OLAP business applications and has twenty two heavy-weight select queries and two update queries. Super-linear or linear speed-up was obtained with ParGRES [9] in all queries.

The TPC-H benchmark can be considered a typical, well-behaved application. In our previous work [9] we had to force a non-uniform valued database to test query load-balancing. In this work, we evaluate ParGRES' performance with a real-world OLAP application, from the Brazilian Institute of Geography and Statistics – IBGE. We focus on analyzing the ParGRES non-intrusive approach of its dynamic adaptive virtual partitioning design. This partitioning along with its load balancing during intra-query parallel processing is critical in the high-performance of OLAP queries.

IBGE is the main provider of data and information about Brazil and is the government agency responsible for Brazil's censuses. IBGE provides for *ad-hoc* query access to the census data through the Statistical Multidimensional Database (BME), which is an OLAP database. Such queries are used by several municipalities and Federative Units to define urban planning. BME is also used in emergency situations such as evaluating the effects of a devastating disaster on local populations. BME is a large database and in our tests, we use a 20 GByte database. We analyzed BME query logs for the last two years to come up with fourteen typical SQL queries. These queries have several joins including fact tables

as well as dimension tables, aggregations, sub-queries, and predicates with all kinds of ranges of selectivity factors.

To evaluate BME queries, all we had to do was to give ParGRES the names of the fact tables and their partitioning attributes, on which we created clustered indexes[1]. We ran BME queries on ParGRES by using a PC cluster with up to 64 nodes from Grid'5000 [4] and PostgreSQL [13] as the sequential DBMS running on each node. We replicated the BME database to be virtually partitioned during query execution. We performed several experiments with ParGRES and we obtained linear and almost always super-linear speedup on queries frequently issued to the BME census. We noticed that, by using only a 4 node PC cluster, queries that take about 15 minutes drop to just half a minute. Since queries are *ad-hoc*, we did not perform any fine tuning nor any optimization on PostgreSQL. Application migration costs and investments on this solution are negligible, being restricted to acquiring an off-the-shelf PC cluster. These results can be further improved if caching and other optimizations are used. We believe the same results can be obtained with other OLAP applications such as TPC-H and BME. Particularly in Brazil, the government has many other OLAP databases that could benefit from ParGRES. Extensive results show that ParGRES has proved to be a very cost-effective alternative solution for OLAP query processing in real scenarios.

This paper is organized as follows. Section 2 shows ParGRES' main features. Section 3 describes the BME census database used in our experiments. Section 4 discusses experimental results. Related work is presented in Section 5 and Section 6 concludes.

## 2. ParGRES

ParGRES [9, 12] is a high-performance database cluster middleware specially developed for OLAP query processing. By exploiting inter- and, in particular, intra-query parallelism, it significantly speeds up the execution of heavy-weight queries, typical from OLAP applications. Fig. 1 shows the general architecture of a DBC with ParGRES.

Before using ParGRES, it is necessary to set the number of nodes to process queries in parallel. When ParGRES receives an SQL query from client applications, it parses and analyzes it in order to determine if it can be processed through intra-query parallelism (following the guidelines specified in [1]). If this is the case, all nodes set in ParGRES are involved in the parallel processing. It re-writes the query in sub-queries to be executed by the DBMS of the database cluster. Sub-queries are generated by using the technique called Virtual Partitioning (VP) [1]. Basically, it consists in rewriting the original query by adding range predicates to it, which originates as many sub-queries as the number of nodes available. Then, each node receives a different sub-query. If the database is fully replicated at all nodes, ParGRES can assign any sub-query to any node. Let us give an example with query Q on table part:

1 Clustered indexes are typically found in most DBMS. They order (cluster) tuples physically according to the index.

```
Q:      select sum (price)
        from part
        where category = 'Y';
```

In order to implement VP, Q would be rewritten as follows:

```
Qi:     select sum (price)
        from part
        where category = 'Y'
        and pid >= :v1 and pid < :v2;
```

In the example, `pid` is chosen as the *virtual partitioning attribute* (VPA) for table `part`. During query rewriting, a different pair of values (v1, v2) is assigned to each node. Similarly to Akal *et al.* [1], ParGRES does that by equally dividing the total value range currently assigned to `pid` by the number of nodes that are going to process the query and giving each one exactly one sub-range. If, for example, `pid` current range is [1, 1000] and four nodes $n_j$ (j = 1 to 4) are configured in ParGRES, to process Q, node $n_1$ receives v1 = 1 and v2 = 251, $n_2$ receives v1 = 251 and v2 = 501, and so on. At each node the local sequential DBMS processes the sub-query on the specified exclusive virtual partition of the table `part`. This way, a database that is not physically partitioned can still be processed through intra-query parallelism. VP is adopted by ParGRES but only to perform its initial query rewriting.

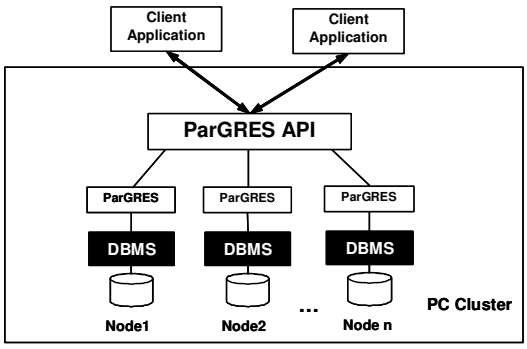


**Fig. 1.** ParGRES Database Cluster Architecture

In ParGRES, when a node first receives a VP sub-query with its value range to VPA, it does not directly submit it to the local DBMS. At each node, ParGRES locally subdivides the value range generating "smaller" virtual partitions, i.e. sub-queries. The idea is to have more flexibility on the load each node has to process. Since VP is based on table replication any node can process any sub-query. The approach of equally dividing the initial range by the number of nodes relies on the absence of data skew. Otherwise once a DBMS receives its sub-query the DBC can no longer reassign part of it to another node to do load balancing.

At the node level, ParGRES adopts a variation of VP called Adaptive Virtual Partitioning (AVP) proposed by Lima *et al.*[7]. The definition of the value range of the VPA is dynamically generated by adapting it to the processing response time of the nodes. When a node receives its sub-query, it subdivides the predicate value range into smaller ranges. Then it starts processing the range by sequentially submitting as many sub-queries as it takes to the local DBMS. Range sizes are continuously adapted by AVP in order to avoid full table scans (more details in [8]). Such an approach of having many sub-queries per node allied to database replication allows for dynamic load balancing. ParGRES redistributes the load by reassigning virtual partition ranges between nodes during query execution, as described in [8]. When a node finishes processing its sub-query, it sends messages to other nodes offering help to process their sub-queries. Due to replication, a node that needs help can reassign part of its unprocessed attribute range to this idle node.

VP requires an extra query processing phase to compose the final result from the partial ones produced by the sub-queries. ParGRES employs HSQLDB [5], a fast lightweight open-source DBMS, to perform result composition. For example, for processing Q, ParGRES would create a temporary table on HSQLDB as follows:

```
create table tempResult (sprice double);
```

Each sub-query result would then be inserted into `tempResult`. When all nodes are done, ParGRES performs final result composition by issuing the following query to HSQLDB:

```
select sum (sprice) from tempResult;
```

The final result is sent to the client application and `tempResult` is then discarded.

ParGRES is fully distributed and it has local components running on each node. Result composition is a two-phase process: in the first phase, each node uses its local HSQLDB instance to perform local result composition; when all nodes are done, their final results are sent to global ParGRES components that use a global HSQLDB the same way to obtain the final result.

ParGRES also supports data updates, requiring no read-only query processing interruption to process them. As data updates are beyond the scope of this work, we refer the reader to [9], to find more detail.

## 3. Population and Housing Census 2000 and BME *Ad-hoc* Queries

IBGE is responsible to support the demands of several different segments of civil society, as well of other governmental institutions at federal, state and municipal levels [2].

Population and Housing Censuses are the main sources of information about the living conditions of the population in each one of the municipalities and localities of Brazil. The censuses produce basic information for the formulation of public policies and for the decision-making processes of private or governmental investments [2]. The data of the survey we used in our tests was collected in the period of August, 1st to November, 30th, 2000, encompassing 5,507 Brazilian municipalities created and installed up to that date. All information from this Census are disseminated and organized in two data collections: Data of the Universe and Data of the Sample, and each data collection in three themes: Housing Units, Persons and Households.

Through the Internet, IBGE makes its survey results available as dynamic pages, downloadable files and online tools to access data. One of these online tools is the Statistical Multidimensional Database - BME.

BME [11] is an OLAP database developed by IBGE and its content is formed by microdata[2] from IBGE's surveys and the metadata associated to these data. This application is intended for researchers who need to generate their own statistical information using microdata and for professionals involved in planning tasks and decision-making processes. BME allows its users to perform *ad-hoc* queries in a database that has more than one billion tuples.

Population and Housing Census 2000 is available in BME and it is one of the greatest surveys in this database, with about 250 millions tuples. It is also the survey with the greatest number of submitted queries by BME users.

The Data of the Sample specifications in BME contains 84 dimension tables, including one time dimension and nine spatial dimensions, and three fact tables: DOMI (housing units data), PESS (persons data) and FAMI (households data). Time dimension refers to the period of the survey and spatial dimensions refer to the Brazilian territorial area.

Fig. 2 shows the relationship between fact tables and time and spatial dimension tables. It presents time dimension (T004), spatial dimensions (G000, G031, G032, G033, G034, G035, G036, G037, G039 and G042) and fact tables (DOMI, PESS and FAMI). Fig. 2 also shows the cardinality of each table. Time and spatial dimensions are related to all fact tables through their foreign keys.

Fig. 3 shows the relationship between fact tables. DOMI is related to the others fact tables, PESS and FAMI, and FAMI is related to PESS, through its primary key.

Fig. 4 illustrates the relationship between DOMI and other dimension tables, excluding the time and the spatial dimensions. There are 24 dimensions, in this case. Their cardinalities are: |M003| = 12, |M075| = 3, |M078| = 5, |M102| = 4, |M103| = 7, |M104| = 4, |M105| = 4, |M106| = 4, |M109| = 8, |M115| = 8, |M116| = 8, |M128| = 6, |M129| = 8,

2 Microdata are nonagregated data about individual objects (persons, housing units, householders, etc.) collected by statistical surveys, that is, the microdata is a record set about one object.

|M208| = 2, |M209| = 12, |M233| = 5, |M270| = 3, |M272| = 10, |M273| = 11, |M274| = 7, |M275| = 11, |M276| = 11, |M277| = 11 and |M278| = 11.

Fig. 5 shows the relationship between FAMI and seven dimension tables, excluding the time and the spatial dimensions. Their cardinalities are: |M290| = 16, |M291| = 17, |M292| = 17, |M293| = 8, |M295| = 13, |M296| = 16, |M297| = 13.

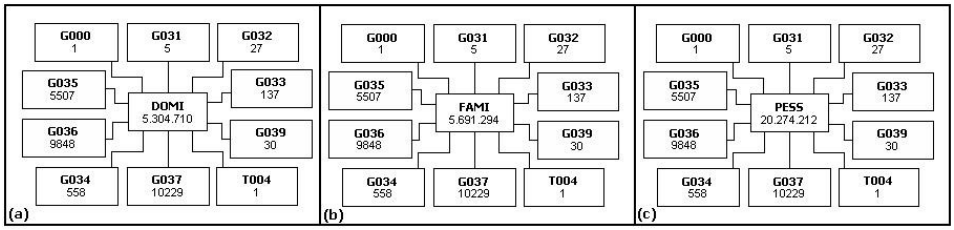


**Fig. 2.** Relationship between fact tables and dimension tables

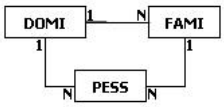


**Fig. 3.** Relationship between fact tables

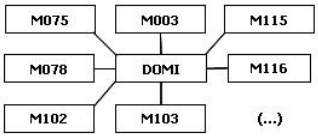


**Fig. 4.** Relationship between DOMI and dimension tables

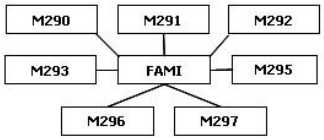


**Fig. 5.** Relationship between FAMI and dimension tables

Fig. 6 illustrates the relationship between PESS and other dimension tables, excluding the time and the spatial dimensions. There are 42 dimensions, in this case. Their cardinalities are: |M159| = 7, |M167| = 4, |M298| = 13, |M300| = 2, |M301| = 13, |M302| = 21, |M306| = 144, |M307| = 54, |M308| = 6, |M309| = 7, |M311| = 3, |M314| = 8, |M315| = 5510, |M320| = 3, |M321| = 5, |M322| = 15, |M323| = 11, |M324| = 12, |M325| = 13, |M326| = 5, |M327| = 63, |M330| = 5, |M331| = 7, |M332| = 7, |M333| = 4, |M334| = 513, |M338| = 224, |M341| = 10, |M342| = 7, |M343| = 5, |M348| = 15, |M355| = 3, |M361| = 8, |M4210| = 98, |M4219| = 260, |M4230| = 99, |M4239| = 261, |M4276| = 5605, |M4279| = 232, |M4300| = 21, |M4354| = 94 and |M4511| = 3.

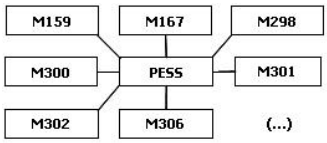


**Fig. 6.** Relationship between PESS and dimension tables

## 4. Experimental Evaluation

In this section, we describe our experimental evaluation obtained by running BME OLAP queries on a database cluster managed by ParGRES. Section 4.1 describes our experimental setup and Section 4.2 analyzes the results we obtained.

### 4.1. Experimental Setup

The experiments were executed on a 64-node cluster of Grid'5000 [4]. Grid'5000 is a French project that provides a large-scale reconfigurable grid infrastructure to support distributed and parallel experiments. Each node has two Intel Xeon 2.3 GHz processors, 4 GB RAM and 160 GB HD. A PostgreSQL 8.2.4 DBMS [13] instance, running on Debian Linux version sid for amd64, is used to manage the Population and Housing Census (BME) database at each node.

The BME database was generated according to specifications from the Statistical Multidimensional application [11]. It contains the dimension and fact tables described in the previous section. We created clustered indexes for the primary keys of each fact table, as required by VP. The total database size is 20 GB and it was fully replicated on the 64 cluster nodes used during our experiments.

Due to *ad-hoc* BME OLAP queries, we analyzed past query loads and selected some typical queries for our experiments. More specifically, we analyze ParGRES performance while executing 14 queries with different complexity levels as shown in Table 1.

All queries use at least one fact table and two out of the dimension tables: one from the time dimension and one from the spatial dimensions. Also, all queries perform at least two aggregations. Queries Q3 and Q6 use only two dimensions (joined to the fact table) and perform four aggregations; Q3 does not have any predicate, while Q6 has a 33.17% selective predicate. Queries Q1, Q2, Q4, Q5, Q7 and Q8 perform joins between a fact table and other dimensions besides the time and the spatial ones. Q1, Q2 and Q4 do not have any predicate. Q4 performs 3 aggregations. Q5 has an 83.95% selective predicate. Q7 and Q8 have high selective predicates, retrieving 2.92% and 3.28% of tuples and perform 4 aggregation functions. Queries Q9, Q10, Q11 and Q13 use two fact tables. Q12 and Q14 use all fact tables in addition to dimension tables. Only Q9 and Q13 do not have predicates, while Q10, Q11, Q12 and Q14 have high selective predicates, retrieving 2.32%, 2.56%, 6.87% and 1.13% of tuples, respectively. Q12 performs 6 aggregations. Q7, Q8, Q10 and Q11 perform a join between a fact table and a relatively large dimension. All queries perform grouping operations. Table 1 summarizes their main characteristics.

**Table 1.** Main characteristics of each query

| Query | Number of aggregations | Predicate | Number of Dimension Tables | Name(s) of Fact Table(s) |
|---|---|---|---|---|
| **Q1** | 2 | N | 3 | DOMI |
| **Q2** | 2 | N | 3 | PESS |
| **Q3** | 4 | N | 2 | PESS |
| **Q4** | 3 | N | 3 | PESS |
| **Q5** | 2 | Y | 3 | PESS |
| **Q6** | 4 | Y | 2 | PESS |
| **Q7, Q8** | 4 | Y | 3 | PESS |
| **Q9** | 2 | N | 4 | DOMI, PESS |
| **Q10, Q11** | 4 | Y | 4 | DOMI, PESS |
| **Q12** | 6 | Y | 3 | DOMI, PESS, FAMI |
| **Q13** | 2 | N | 3 | DOMI, PESS |
| **Q14** | 2 | Y | 6 | DOMI, PESS, FAMI |

### 4.2. Experimental Results

In this section, we present the results obtained during our speedup experiments. Table 2 shows the execution times (in seconds) obtained for each BME query while adding cluster nodes (from 1 to 64 nodes).

Table 2 provides evidence of the excellent results obtained, which are better shown in Fig. 7, that we split in two figures to improve legibility. We present the normalized

execution time for each query in Fig. 7 (a) and (b). Normalization was obtained by dividing each query execution time by the highest execution time of its associated query, i.e., the sequential execution of the query. With the sole exception of Q1, which is already quite fast, all other queries have super-linear speedup for all node configurations, as shown in Fig. 8. The speedup curves in Fig. 8 (a) and (b) are shown using log scale for y-axis and the x-axis represents the number of nodes. Fig. 8 also shows the linear speedup curve plotted with the thickest line style.

**Table 2.** Execution times (in seconds) per number of nodes

| Query | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
|---|---|---|---|---|---|---|---|
| Q1 | 31.90 | 13.59 | 9.00 | 7.86 | 2.61 | 2.50 | 2.36 |
| Q2 | 217.41 | 113.30 | 24.41 | 17.85 | 13.70 | 7.18 | 2.50 |
| Q3 | 217.47 | 107.60 | 16.81 | 13.96 | 7.94 | 5.78 | 2.19 |
| Q4 | 217.90 | 101.76 | 16.89 | 12.88 | 12.36 | 4.79 | 2.24 |
| Q5 | 217.44 | 88.77 | 10.70 | 10.38 | 9.12 | 4.47 | 1.69 |
| Q6 | 217.62 | 92.48 | 11.45 | 11.42 | 7.44 | 1.91 | 1.75 |
| Q7 | 217.76 | 85.98 | 12.33 | 10.55 | 5.60 | 1.51 | 1.35 |
| Q8 | 217.67 | 86.19 | 10.08 | 9.01 | 8.35 | 2.30 | 1.81 |
| Q9 | 460.30 | 208.19 | 41.30 | 22.52 | 11.03 | 5.71 | 3.60 |
| Q10 | 586.65 | 244.96 | 16.18 | 10.86 | 8.86 | 5.70 | 1.70 |
| Q11 | 568.54 | 273.66 | 13.35 | 12.74 | 11.54 | 2.30 | 1.82 |
| Q12 | 1.242.22 | 618.85 | 174.95 | 84.45 | 42.25 | 21.91 | 11.43 |
| Q13 | 218.37 | 112.56 | 23.15 | 11.68 | 8.15 | 5.83 | 2.94 |
| Q14 | 1.030.08 | 511.11 | 93.80 | 48.08 | 25.50 | 14.45 | 9.20 |

With 4 nodes, the virtually partitioned database may be fitting in the four nodes main memory, which explains the significant speedup for all queries with just 2 or 4 nodes in Fig 7. However, even after the memory effect, execution time continues to drop significantly up to the 64 nodes. Q12 is the most time consuming query and its execution drops from 21 minutes to 1.5 minute with 8 nodes and to only 11.43 sec with 64 nodes. Even though some queries present some skew (Q7, Q8, Q10, Q11, Q12 and Q14), it was the same behavior of ParGRES while processing queries with uniform data. Considering that several time-consuming queries are submitted one after the other, often with one depending on the result of the other, intra-query parallelism of ParGRES can significantly reduce the overall time needed for the decision making process. This is crucial in many situations such as disease control measures based on census dimensions and fact tables.

## 5. Related Work

Besides ParGRES, there are several DBC solutions designed to improve query processing performance through parallel processing techniques, like PowerDB [14], C-JDBC [3], Apuama [10] and Sequoia [18], which is a follow up of C-JDBC. Only ParGRES and

PowerDB provide for intra-query parallelism, but PowerDB presents severe limitations in the presence of load unbalancing between nodes during query processing.

PowerDB [14] is a DBC middleware that uses full database replication as physical data organization scheme to obtain parallel query processing. Different middleware solutions have been proposed in PowerDB for OLAP and OLTP query processing. With respect to OLAP, Akal *et al.* [1] have originally proposed VP to obtain intra-query parallelism. However, VP has limitations as it relies on a uniform data value distribution for the VPA to achieve high-performance during query processing. Its "one sub-query per node" approach is also an issue as it prevents dynamic load balancing due to black-box DBMS. In our previous work [9] we show that our dynamic AVP outperforms VP for all TPC-H queries. Particularly, in the presence of data skew, the static features of using just VP imposed poor performance results, often with slow down factors.

Röhm et al. [15] propose Hybrid Partitioning (HP), a physical database design approach that avoids full database replication by equally fragmenting the fact tables and allocating each fragment to a different cluster node. The dimensions are fully replicated between nodes. HP makes it possible to implement intra-query parallelism during OLAP query processing. However, if load unbalancing occurs during query execution, data transfers must be made in order to redistribute load. In fact, no dynamic load balancing policy is proposed by the authors. Other works were proposed in the context of PowerDB [16, 17] but they do not employ intra-query parallelism thus not reducing the execution time of individual queries, which is imperative for speeding up decision making process. Finally, PowerDB is not open-source nor is available for download.

C-JDBC is an open-source DBC middleware designed by Cecchet *et al.* [3] just as ParGRES. However, it focuses on OLTP and e-commerce applications. It uses full and partial database replication to obtain inter-query parallelism. Intra-query parallelism is not supported. C-JDBC shows good performance for the TPC-W Benchmark [21]. In particular, partial replication is better than full replication because of data updates, typical from e-commerce applications. However, without intra-query parallelism, C-JDBC is not adequate to OLAP query processing. It is more suited to applications that issues fast concurrent queries, such as TPC-C [19] and TPC-W [21].

Sequoia [18] is the sequence of the C-JDBC project and is a solution that offers clustering, load balancing and failover services for any database. Sequoia implements the concept of Redundant Array of Inexpensive Databases (RAIDb) (just as C-JDBC) to provide partial replication in order to tune the degree of replication of each database. It also improves performance using fine grain query caching and transparent connection pooling. Furthermore, Sequoia offers support for clusters of heterogeneous database engines. Just as C-JDBC, it fits better for OLTP and e-commerce applications.

Apuama is proposed by Miranda *et al.* [10] to add intra-query parallelism to C-JDBC. Intra-query parallelism is implemented through static VP and full database replication, thus achieving good results. However, it does not properly deals with the DBMS as a black-box component as it uses DBMS-specific commands to force the use of clustered indexes during query processing in order to have good performance with VP. In addition, as PowerDB, it is also very sensitive to skew, since it cannot do dynamic load-balancing.

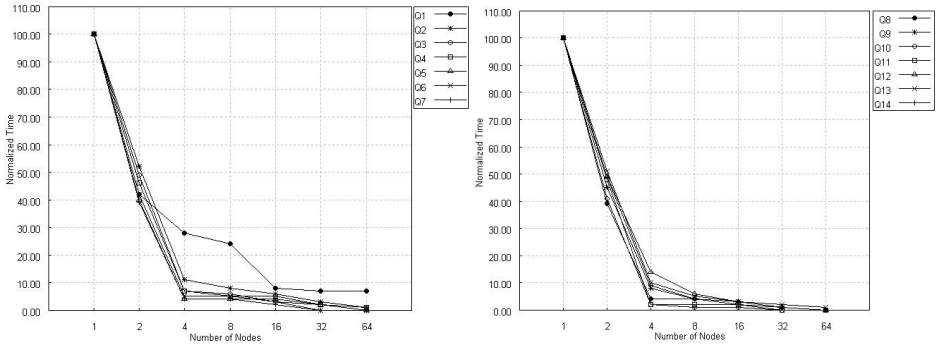


**Fig. 7.** Query execution time experiments: (a) results for Q1-Q7; (b) results for Q8-Q14

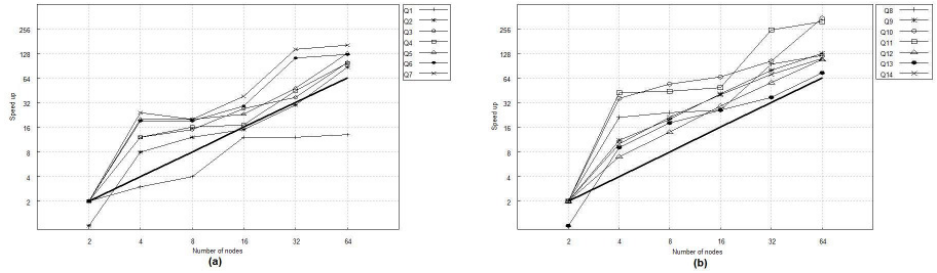


**Fig. 8.** Query speedup: (a) results for Q1-Q7; (b) results for Q8-Q14

## 6. Conclusions

In this paper, we described an evaluation of ParGRES database cluster middleware in a real-world scenario. We have show experiments with Brazilian official census database (Population and Housing Census 2000), 14 BME OLAP queries using a 64-node cluster of the Grid'5000 experimental platform. The Census 2000 database was generated according to BME specifications and it was fully replicated on the 64 cluster nodes used during our experiments. PostgreSQL DBMS was used to manage the database at each node.

Our experiments explored intra-query parallel processing with time consuming typical BME census queries and the behavior of ParGRES open source database cluster middleware by using a real scenario. The results showed that, in almost all cases, ParGRES yields super-linear speedup while adding cluster nodes (from 1 to 64 nodes). With the exception of Q1 which is already quite fast, all other queries get significant super-linear speedup for all node configurations.

Our experimental results show that AVP improves the performance of *ad-hoc* queries in real scenarios, including data skew. These results are very encouraging and make ParGRES a very cost-effective alternative solution for OLAP applications designed by governmental institutions and researchers in general.

In future work, we will address concurrent sequences of queries. The experiments in this paper have focused on queries in isolation. It is necessary to evaluate the system performance by simulating concurrent queries from a few users. Future experiments should also focus on streams of queries as specified in TPC-H which is also typical of BME decision makers.

Currently ParGRES team is developing GParGRES, its extension to grid platforms. Preliminary results on TPC-H also on Grid´5000 show encouraging results [6]. We also plan to evaluate our BME distributed design on top of grids using larger configurations of nodes. However, dynamic load balancing in grids may compromise the speedup obtained with the cluster.

## Acknowledgments

This work was partially funded by CNPq and INRIA. The authors are grateful to the Grid´5000 team for their support.

## References

1. Akal, F., Böhm, K., Schek, H. J.: OLAP Query Evaluation in a Database Cluster: a Performance Study on Intra-Query Parallelism. In: Proceedings of the 6th East-European Conference on Advances in Databases and Information Systems (ADBIS), pp.218-231. Springer, Bratislava, Slovakia, September 3-7 (2002)
2. Brazilian Institute of Geography and Statistics – IBGE, http://www.ibge.gov.br
3. Cecchet, E., Marguerite, J., Zwaenepoel, W.: C-JDBC: Flexible Database Clustering Middleware. In: Proceedings of USENIX Annual Technical Conference, Freenix Track, pp.9-18. Boston, EUA, June (2004)
4. Grid5000 Projects Web Site - Grid5000, http://www.grid5000.fr
5. HSQL Database Engine, http://hsqldb.org/

6. Kotowski, N., Lima, A. A., Pacitti, E., Valduriez, P., Mattoso, M. L. Q.: Parallel Query Processing for OLAP in Grids. Concurrency and Computation. Practice & Experience, http://dx.doi.org/10.1002/cpe.1303 (2008)
7. Lima, A. A. B.: Intra-query parallelism in Database Clusters (*in Portuguese)*. Ph.D. Thesis, COPPE/UFRJ, Rio de Janeiro (2004)
8. Lima, A. A. B., Mattoso, M., Valduriez, P.: Adaptive Virtual Partitioning for OLAP Query Processing in a Database Cluster. In: Proceedings of the 19th Brazilian Symposium on Database Systems (SBBD 2004), pp.92-105. Brasília, Brazil, October 18-20 (2004)
9. Mattoso, M., Zimbrão, G., Lima, A. A. B., Baião. F., Braganholo, V., Aveleda, A., Miranda, B., Almentero, B., Costa, M.N.: ParGRES Middleware for Executing OLAP Queries in Parallel. Technical report, http://pargres.nacad.ufrj.br/Documentos/ES-690.pdf (2005)
10. Miranda, B., Lima, A. A. B., Valduriez, P., Mattoso, M.: Apuama: Combining Intra-query and Inter-query Parallelism in a Database Cluster. In: Currents Trends in Database Technology (EDBT 2006), LNCS, vol. 4254, pp.649-661. Springer, Heidelberg (2006)
11. Multidimensional Statistics Database – BME, http://www.bme.ibge.gov.br
12. ParGRES, http://pargres.nacad.ufrj.br/
13. PostgreSQL, http://www.postgresql.org
14. POWERDB, http://www.dbs.ethz.ch/archive/index.html
15. Röhm, U., Bohm, K., Schek, H-J.: OLAP Query Routing and Physical Design in a Database Cluster. In: Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology (EDBT 2000), pp.254-268. Springer-Verlag, London, UK, March 27-31 (2000)
16. Röhm, U., Bohm, K. Schek, H-J.: Cache-Aware Query Routing in a Cluster of Databases. In: Proceedings of the 17th International Conference on Data Engineering (ICDE 2001), pp.641-650. IEEE Computer Society, Heidelberg, Germany, April 2-6 (2001)
17. Röhm, U., Böhm, K., Scheck, H.-J., Schuldt, H.: FAS - A Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components. In: Proceedings of the 28th International Conference on Very Large Databases Conference (VLDB 2002), pp.754-765. Hong Kong, China, August 20-23 (2002)
18. Sequoia Project, http://sequoia.continuent.org/HomePage
19. TPC Benchmark C, http://www.tpc.org/tpcc/
20. TPC Benchmark H, http://www.tpc.org/tpch/
21. TPC Benchmark W, http://www.tpc.org/tpcw/default.asp
22. Valduriez, P.: Parallel Database Systems: open problems and new issues. In: International Journal on Distributed and Parallel Databases, vol.1, n.2, pp.137-165. April (1993)