A Parallel Algorithm to Compute Data Synopsis

Carlo DELL'AQUILA, Francesco DI TRIA, Ezio LEFONS, and Filippo TANGORRA
Dipartimento di Informatica
Università degli Studi Bari "Aldo Moro"
via Orabona 4, 70125 Bari
ITALY
{dellaquila, francescoditria, lefons, tangorra}@di.uniba.it

Abstract: - Business Intelligence systems are based on traditional OLAP, data mining, and approximate query processing. Generally, these activities allow to extract information and knowledge from large volumes of data and to support decisional makers as concerns strategic choices to be taken in order to improve the business processes of the Information System. Among these, only approximate query processing deals with the issue of reducing response time, as it aims to provide fast query answers affected with a tolerable quantity of error. However, this kind of processing needs to pre-compute a synopsis of the data stored in the Data Warehouse. In this paper, a parallel algorithm for the computation of data synopses is presented.

Key-Words: - data warehouse, approximate query processing, data synopsis, parallel algorithm, message passing interface.

1 Introduction

The core of a Business Intelligence system is represented by a Data Warehouse, which is a repository built to store large volume of data. Such data are obtained by the integration of operational data, coming from heterogeneous data sources (such as relational databases, XML files, Excel documents) adopted in transactional systems. Since a Data Warehouse is used in the decision making process, it must be designed in order to support statistical analyses of data [1, 2]. Moreover, it must allow analyses based on temporal series. For this reason, a Data Warehouse must not only integrate data coming from operational databases but also preserve historical data, accumulating data over time. In this way, it is clear that the cardinality of the tables of the Data Warehouse increases very fast, because records are inserted when the Data Warehouse is fed and never deleted.

Thus, the data stored in a Data Warehouse are used in the On-Line Analytical Processing (OLAP), in order to produce information to be used in the decision making process. OLAP consists of a set of analytical queries, essentially based on statistical functions, and typical OLAP operators, as *drill-down*, *roll-up*, *slice-and-dice*, and *pivoting* [3]. In particular, the statistical functions based on the SQL aggregation and grouping operators require usually a long answer time [4, 5].

As the results of statistical computations are used for business strategic choices, often decisional makers are not interested in exact values but approximate values will suffice. In fact, in this context, it may be more suitable to obtain approximate values quickly, rather than exact values, requiring a high answer time.

Nowadays, there are several systems supporting approximate query answering, based on different methodologies, such as wavelet [6], sampling [7, 8], and graph-based modelling [9]. In spite of the adopted methodology, all these systems share the following process model: (a) calculating the data synopsis and (b) using the calculated data to execute analytical queries [10]. It has been widely verified that these systems are able to produce answers in lower time than traditional systems, with an acceptable percentage of error [11].

In particular, the methodology presented here is based on the analytical data profile [12]. According to this methodology, the data synopsis is represented by a set of computed values, the so-called Canonical Coefficients (CCs), that contain information about the multivariate distribution of the data stored in the Data Warehouse. The computational time to obtain these coefficients is very high. The number of coefficients that must be generated depends on both the approximation function degree (in fact, the CCs are the coefficients of the approximation polynomials) and the number of attributes involved in the computation. Moreover, the computational process needs to scan the entire relations. For these reasons, the computational time depends on (a) the degree of approximation, (b) the number of attributes, and (c) the cardinality of the relation.

As the generation time of the data synopsis is a standard criterion to evaluate approximate query answering methodologies, here we investigate an extension of our analytic method by designing a

parallel algorithm in order to decrease the computational time needed to generate the CCs.

Thus, the paper aims to present a parallel algorithm able to compute the CCs in a distributed way and to report the evaluation tests. The experimental setup is devoted to show the effective decreasing of the computational time in reference to the used resources.

The paper has the following structure. Section 2 presents the main architecture of the system. Section 3 points out the methodology used to compute the CCs and introduces the related parallel version of the algorithm. Section 4 shows the parallel architecture used for the experimentation. Section 5 discusses the obtained experimental results. Section 6 contains an optimized version of the parallel algorithm. Finally, Section 7 reports our conclusions.

2 Approximate Query Answering Systems

The Approximate Query Answering System is an analytical tool that allows business decision makers to obtain fast approximate answers to complex database queries. As a counterpart, such answers may be affected with small errors. Since the analytical processing is usually very complex and commonly consists of aggregate functions on large relations, the extraction of information from data is very slow. In these cases, decisional makers could prefer to obtain and to use approximate but reliable values.

As an example, a decision maker may be interested in determining the best employee of his/her business company in the last year. This information requires computing the number of products sold by each employee during the last year. Thus, if the best employee sold the 51.5% of the total products, then the decision maker could tolerate 50% as business answer, since this value represents an optimal approximation of the real answer and does not falsify the final information, as the approximate answer is affected with a very low error and it can be returned quickly.

Here, we focus on Approximate Query Answering Systems performing a data reduction. This always happens when utilizing methodologies based on polynomial approximation, sampling, and wavelets. In fact, these systems are able to provide approximate answers using small and pre-computed data synopses, obtained by suitably reducing the data stored in the database. Usually, the database used as a data source in Decision Support Systems is a Data Warehouse [13].

The data synopsis is represented by a set of coefficients, used to calculate aggregate functions. Of course, the methods to compute aggregate values are overridden. That is, the aggregate functions (such as sum, average, and count) are processed according to *ad hoc* algorithms. Further features of these systems include accuracy bounds, without making *a priori* assumptions on the data distribution. At last, these systems can often include components that allow decision makers to obtain exact values, in presence of critical factors or when the total precision is needed.

In Figure 1, there is depicted a high-level architecture that shows a Decision Support System based on both a Data Warehouse, which processes datasets in order to provide exact values, and an Approximate Query Answering System, which is able to compute fast and approximate answers, using a set of coefficients representing a synopsis of the data stored in the Data Warehouse.

For the sake of simplicity, on the basis of such an architecture, the decision maker can define simple and/or complex business indicators and, then, s/he is allowed to obtain both fast and approximate query responses or the exact value against a higher response time.

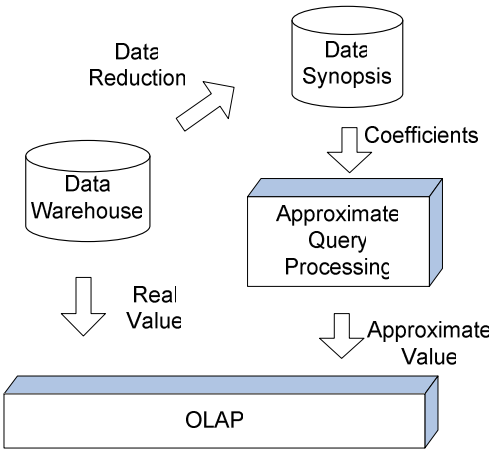

Fig. 1. Architecture of a Decision Support System.

3 Canonical Coefficients Methodology

The method consists of using a polynomial series to approximate the multivariate data distribution function of m attributes X_1, X_2, …, X_m. The polynomial is the Legendre orthogonal polynomial series whose coefficients provide synthetic information about the multivariate data distribution.

Let $R(X_1, X_2, \ldots, X_m)$ be a relation of cardinality n. We assume $dom(X_j) = [a_j, b_j]$, for each $j = 1, 2, \ldots, m$. That is, the domain of attribute X_j is a numeric (real) interval.

So, we define $D = [a_1, b_1] \times \ldots \times [a_m, b_m]$.

Finally, let *pdf*(*x*) be the probability density function of R. We denote with *g*(*x*) its polynomial approximation up to degree *d*.

Since the Legendre orthogonal polynomials are defined on the interval [-1, 1], each value $y \in dom(X_j)$ is suitably mapped to the corresponding value $y' \in [-1, 1]$.

Then, $\forall x = (x_1, x_2, \ldots, x_m) \in R$, it results that:

$$g(x) = \frac{1}{2^m} \sum_{i=0}^{d} \sum_{\substack{i_1,\ldots,i_m \\ i_1+\ldots+i_m=i}} (2_{i_1}+1)\cdots(2_{i_m}+1) c_{i_1,\ldots,i_m} P_{i_1,\ldots,i_m}(x')$$

,

where:

1. $x \rightarrow x' = (x'_1, x'_2, \ldots, x'_m)$ is the opportune isomorphism from $x \in D$ to $x' \in [-1, 1]^m$,
2. $P_{i_j}(x'_j)$ is the Legendre polynomial of degree i_j,
3. $(i_1, \ldots, i_m)$ is an *m*-tuple of natural numbers such that their sum yields *i*,
4. $P_{i_1, \ldots, i_m}(x') = P_{i_1}(x'_1) \times \ldots \times P_{i_m}(x'_m)$ is the *m*-dimensional Legendre polynomial of degree *i* on the interval $[-1, 1]^m$, and
5. $c_{i_1, \ldots, i_m} = \frac{1}{n} \sum P_{i_1, \ldots, i_m}(x')$ is the mean value of $P_{i_1, \ldots, i_m}(x')$ on the *n* tuples *x* of R.

Therefore, *g*(*x*) is the orthogonal polynomial approximation to *pdf*(*x*) up to degree *d* and the coefficients $\{c_{i_1, \ldots, i_m} \mid i_1+\ldots+i_m = i, i = 0, \ldots, d\}$ carry information in order to represent the *m*-dimensional data distribution of the relation R.

These coefficients are the so-called *Canonical Coefficients* of R and they can be used in order to calculate quickly aggregate functions, such as count, sum, and average, in an approximate way (*cf.*, [12]).

3.1 Generation Algorithm

Let M[*n*, *m*] be a matrix of $n \times m$ numeric values and let x_{ij} be the value of the *i*-th row and *j*-th column of the matrix M.

Let $dom(X_j)$ be the domain of the *j*-th column. Then, Min_j and Max_j denote, respectively, the minimum and the maximum of $dom(X_j)$. Let x'_{ij} be the normalization of x_{ij} in the interval [-1, 1], such that $x_{ij} \in [Min_j, Max_j] \Rightarrow x'_{ij} \in [-1,1]$.

Finally, let *Legendre*(x'_{ij}, *d*) be the Legendre function, calculated on x'_{ij} according to the degree *d*. This function returns a floating point value.

The following pseudo-code describes the algorithm ALG to generate the CCs.

Input
n = {number of rows}
m = {number of columns}
dg = {approximation degree}
M = {numeric matrix}

Output
CC // vector of the Canonical Coefficients

Pseudo-code of the algorithm ALG
for d = 0 to dg
for each $(d_0, d_1, \ldots, d_{m-1})$ *such that*
$(d_0 + d_1 + \ldots + d_{m-1}) = d$
do
for i = 0 to n−1
$pr = Legendre(x'_{i0}, d_0) \times Legendre(x'_{i1}, d_1) \times \ldots \times Legendre(x'_{im-1}, d_{m-1})$
end for
$CC_z = average(pr)$
increment index z
end do
end for

where *z* = 1 to $\binom{dg+m}{dg}$.

Example 1. Let $R(X_1, X_2)$ be the relation shown in Table 1. For simplicity, let $d = 2$ the degree of approximation, $m = 2$ the number of attributes, and $n = 4$ the number of rows.

Here, $Min_1 = 1$, $Max_1 = 7$, $Min_2 = 3$, and $Max_2 = 11$. Then, each value of the relation R is suitably normalized in the interval [-1, 1], as shown in Table 2.

R	
X_1	**X_2**
1	3
3	11
2	9
7	5

Table 1. Instance of the relation R.

R′	
X'_1	**X'_2**
-1	-1
-0.33	1
-0.66	0.5
1	-0.5

Table 2. The normalized relation R' of R.

Now, the algorithm ALG computes the following quantities.

For $d = 0$.

$CC_1 = (Legendre(-1, 0) \times Legendre(-1, 0) + Legendre(-0.33, 0) \times Legendre(1, 0) + Legendre(-0.66, 0) \times Legendre(0.5, 0) + Legendre(1, 0) \times Legendre(-0.5, 0)) / 4$

For $d = 1$.

$CC_2 = (Legendre(-1, 1) \times Legendre(-1, 0) + Legendre(-0.33, 1) \times Legendre(1, 0) + Legendre(-0.66, 1) \times Legendre(0.5, 0) + Legendre(1, 1) \times Legendre(-0.5, 0)) / 4$

$CC_3 = (Legendre(-1, 0) \times Legendre(-1, 1) + Legendre(-0.33, 0) \times Legendre(1, 1) + Legendre(-0.66, 0) \times Legendre(0.5, 1) + Legendre(1, 0) \times Legendre(-0.5, 1)) / 4$

For $d = 2$.

$CC_4 = (Legendre(-1, 2) \times Legendre(-1, 0) + Legendre(-0.33, 2) \times Legendre(1, 0) + Legendre(-0.66, 2) \times Legendre(0.5, 0) + Legendre(1, 2) \times Legendre(-0.5, 0)) / 4$

$CC_5 = (Legendre(-1, 1) \times Legendre(-1, 1) + Legendre(-0.33, 1) \times Legendre(1, 1) + Legendre(-0.66, 1) \times Legendre(0.5, 1) + Legendre(1, 1) \times Legendre(-0.5, 1)) / 4$

$CC_6 = (Legendre(-1, 0) \times Legendre(-1, 2) + Legendre(-0.33, 0) \times Legendre(1, 2) + Legendre(-0.66, 0) \times Legendre(0.5, 2) + Legendre(1, 0) \times Legendre(-0.5, 2)) / 4$

... □

3.2 Parallel Algorithm

In this Sub-section, we introduce the parallel version of the algorithm ALG of Sub-section 3.1.

The parallel algorithm is based on the *divide et impera* approach, according to which each node elaborates a subset of the data stored in the relation. The final result is computed by the root node, that executes I/O operations and collects partial results, calculated in distributed way.

The final result is computed by applying the "additive property" of the CCs (*cf.*, [12]).

This property states that

$$\mathrm{CC}_i = \frac{\sum_{j=1}^{t} N_j \times \mathrm{CC}'_{ij}}{\sum_{j=1}^{t} N_j},$$

where CC'_{ij} is the coefficient of a vector, calculated on a relation of cardinality N_j and t is the number of vectors, whereas each vector is computed by a node.

In the parallel algorithm, first, each node applies locally the algorithm for the generation of CCs. Second, when the computation is ended on each node, the root node applies the additive property. Finally, the last step executed by the root node is the computation of the average value of each coefficient of the vector.

Without loss of generality, we assume that the number of rows n is a multiple of the number of nodes t. If it is not so, we can easily calculate the modulus of the division of n by t, and assign the remaining rows to one of the nodes, usually the root one.

The number of generated CCs (*i.e.*, the cardinality of the vector of the CCs) depends on both the degree of approximation and the number of fields of the chosen relation, but not on the number of rows. The following pseudo-code describes the parallel algorithm.

The functions used for the inter-process communication are:

- *send*(*CC*, $node_i$), which means that the current node sends the vector storing the CCs to the $node_i$, and
- *receive*(*CC*, $node_i$), which means that the current node reads the data sent from $node_i$ and store them in the vector CC.

Input

t = {number of nodes}
n′ = {number of rows} / {number of nodes} = n / t
m = {number of columns}
dg = {approximation degree}
M = {sub-matrix of data}
w = {cardinality of each vector}

Output

CC // vector of Canonical Coefficients, with partial results

Pseudo-code of the parallel algorithm PALG

// generation of the coefficients
for d = 0 to dg
for each $(d_0, d_1, \ldots, d_{m-1})$ *such that*
$(d_0 + d_1 + \ldots + d_{m-1}) = d$
do
for i = 0 to n′−1
$pr = Legendre(x'_{i0}, d_0) \times Legendre(x'_{i1}, d_1) \times \ldots \times Legendre(x'_{im-1}, d_{m-1})$
end for
$CC_z = pr$
increment index z
end do
end for
// starting inter-process communication
if node identifier <> 0 then
// it is not the root node
send(*CC*, $node_0$)

else
// it is the root node
for $q = 1$ *to* $t-1$
receive(CC', $node_q$)
for $i = 1$ *to* w
$CC_i = CC_i + CC'_i$
end for
end for
for $i = 1$ *to* w
$CC_i = CC_i / n$
end for
end if.

The limit of algorithm PALG is that the nodes need to be synchronized among themselves. This happens when each node has finished its own computation and the root node needs to gather partial results, before applying the additive property.

Example 2. This example recalls the previous Example 1. For simplicity, let us suppose to execute the parallel algorithm by involving only two processes. Moreover, in this example, Min_1, Max_1, Min_2, and Max_2 are global values, shared among all the nodes. The first two rows of the relation R are assigned to the root node, identified by the number 0, and the last two rows are assigned to the second node, a process identified by the number 1. Each node calculates its own vector of the CCs.

On the root node, the algorithm works in the following manner.

For $d = 0$.
$CC_1 = Legendre(-1, 0) \times Legendre(-1, 0) + Legendre(-0.33, 0) \times Legendre(1, 0)$

For $d = 1$.
$CC_2 = Legendre(-1, 1) \times Legendre(-1, 0) + Legendre(-0.33, 1) \times Legendre(1, 0)$
$CC_3 = Legendre(-1, 0) \times Legendre(-1, 1) + Legendre(-0.33, 0) \times Legendre(1, 1)$

For $d = 2$.
$CC_4 = Legendre(-1, 2) \times Legendre(-1, 0) + Legendre(-0.33, 2) \times Legendre(1, 0)$
$CC_5 = Legendre(-1, 1) \times Legendre(-1, 1) + Legendre(-0.33, 1) \times Legendre(1, 1)$
$CC_6 = Legendre(-1, 0) \times Legendre(-1, 2) + Legendre(-0.33, 0) \times Legendre(1, 2)$
...

In this case, note that the average is not calculated during the computation of the CCs.

After the computation has finished, node 0 and node 1 need to synchronize themselves, such that node 1 must execute a *send* inter-process communication primitive, while node 0 must execute a *receive* operation. In this way, root 0 elaborates two vectors of CCs, each one containing partial results.

At this point, the root node can apply the additive property and calculate the average value on each value of the vector of the CCs.

The final vector is obtained so:

$CC_1 = (CC_1 + CC'_1) / 4$
$CC_2 = (CC_2 + CC'_2) / 4$
$CC_3 = (CC_3 + CC'_3) / 4$
$CC_4 = (CC_4 + CC'_4) / 4$
$CC_5 = (CC_5 + CC'_5) / 4$
$CC_6 = (CC_6 + CC'_6) / 4$
... □

4 Parallel Architecture

Each node of the parallel architecture is a 2-processor computer, with 2 GHz and 512 MB RAM. The network topology is a star-network (*see*, Figure 2).

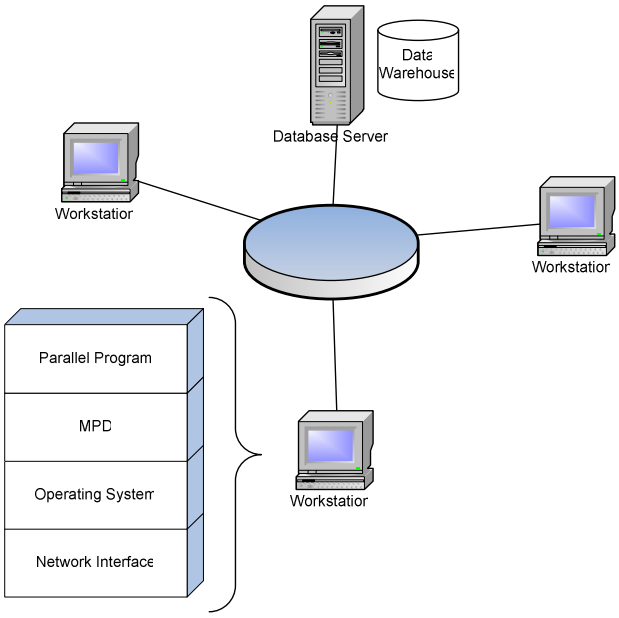

Fig. 2. Layers of the parallel architecture.

The root node is represented by a Database Server, which manages the Data Warehouse. Each node can access the Data Warehouse by querying the Database Server. The Database Server is MySQL 5.1, and the communication between a client and the Database Server is based on the Open Database Connectivity (ODBC) protocol. The client executes a query like the following, in order to load its own sub-matrix of data: "*select … from … limit h, k;*", where *h* and *k* are integer values, representing

the index of the starting row and the number of rows to be retrieved, respectively. For example, the query "*select * from sales limit* 0, 10*"* returns a recordset of ten records, starting from the first row of the *sales* table. On each node, a software tool for parallel computing has been installed. The chosen library for parallel computing is MPICH [14], an open source implementation of MPI [15, 16].

MPI is a software library commonly used to build Boewulf architecture [17], which is a class of computer clusters originally developed by Thomas Sterling and Donald Becker at NASA. Nowadays, Beowulf architectures are widely used all over the world, mainly in academic environments, because they are high-performance parallel computing architectures based on inexpensive personal computer hardware. In fact, a Beowulf architecture is a group of usually identical PC computers running a minimal version of an open source operating systems.

Therefore, each node of our architecture is characterized by the same hardware/software configuration, except for the Database server, where it is installed also the Database Management System. At the bottom level of this configuration, there is the network interface, constituted by an Ethernet interface and the TCP/IP network protocol. The second level is constituted by Windows XP Pro operating system. The third level is constituted by the MPD server process, that is, the program installed by MPICH. This server must be started on each node and its aim is to manage the inter-process communication among the nodes involved in the computation. On the top, there is the parallel program PALG. In this case, the parallel program is a C-program that implements the algorithm explained in Sub-section 3.2.

The MPICH library is able to realize MIMD (Multiple Instructions Multiple Data) architecture, where each node executes different code, using different set of data. Usually, the discrimination is based on a natural number identifying each node. The root node is always identified by the number 0, the others in cascade.

Example 3. This example recalls Example 2 of the previous Section. First, each node establishes an ODBC connection with the Database server. Second, each node executes an SQL query to get data. In order to load its own sub-matrix of data, the node 0 executes the query "*select* X_1, X_2 *from* R *limit* 0, 2", while the node 1 executes the query "*select* X_1, X_2 *from* R *limit* 2, 2". □

4.1 Mapping function

In this Sub-section, we show the mapping function used to assign a sub-matrix of data to each node of the parallel architecture.

Let n be the number of rows of a relation R and let t the number of nodes. First, we need to compute the mean value a of rows to assign to each node:

$$a = \lfloor n / t \rfloor.$$

Then, we compute the modulus b of the division of n by t:

$$b = n \mid t,$$

that is, b represents the further rows to be assigned to the root node. We assume that both the a and b values are shared among the nodes.

Let n_i be the number of rows to be assigned to the node i.

We have that

$$n_0 = a + b, \text{ and}$$
$$n_j = a,$$

for $j = 1, ..., t - 1$.

Therefore, the root node has to load the first n_0 rows of the relation R. The SQL query generated at the root is *"select* X_1, X_2 *from* R *limit* 0, n_0*"*.

The other nodes have to load only a rows. However, we must indicate the starting row from which a node must load the data. Let s_i be the starting row to be assigned to the node i.

We define

$$s_0 = 0, \text{ and}$$
$$s_j = j \times a + b,$$

for $j = 1, ..., t - 1$.

The SQL query generated at the node j is *"select* X_1, X_2 *from* R *limit* s_j, n_j*"*, for $j = 0, 1, \ldots, t - 1$.

Example 4. Let $n = 100$ and $t = 3$. Then, $a = 33$ and $b = 1$.

So, we have

$n_0 = 33 + 1 = 34$, and

$n_j = 33$, for $j = 1, 2$.

The starting row assigned to the root node is always s_0. On the other hand, we have

$s_1 = 1 \times 33 + 1 = 34$, and

$s_2 = 2 \times 33 + 1 = 67$.

At this point, the root node executes the query *"select* X_1, X_2 *from* R *limit* 0, 34*"*, that allows to load 34 rows starting from the first row (that is, from 0 to 33).

The node 1 executes the query *"select* X_1, X_2 *from* R *limit* 34, 33*"*, that allows to load 33 rows starting from the 34th (that is, from 34 to 66).

The node 2 executes the query *"select* X_1, X_2 *from* R *limit* 67, 33*"*, that allows to load 33 rows starting from the 67th (that is, from 67 to 99).

The mapping function is depicted in Figure 3.

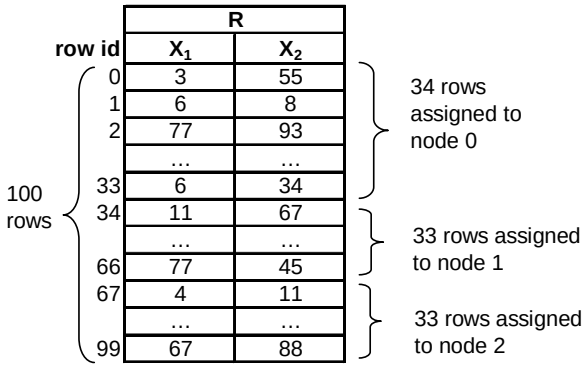

Fig. 3. Mapping function.

5 Experimental results

The table of the Data Warehouse chosen for the generation of the CCs is a relation on the schema *sales*(*product*, *order*, *amount*), whose cardinality is of the order of 400,000 records. Then, the number of attributes involved in the data reduction is 3. Moreover, the approximation degree chosen for this experiment is 27. Therefore, the number of CCs generated by the algorithm is

$$\binom{27+3}{27} = 4060,$$

that represents the number of values that must be transferred among the nodes.

The computational time of the serial program takes 4,556,500 milliseconds. Figure 4 shows the experimental values obtained in the computation of the CCs, in comparison with the expected ones. For the computation of the CCs, the expected time is given by

$$\frac{4{,}556{,}500}{t},$$

for $t = 2, \ldots, 8$, where t is the number of nodes.

For example, if the serial program takes 4,556,500 milliseconds, then the expected time for the parallel program, executed on two processors, is

$$\frac{4{,}556{,}500}{2} = 2{,}278{,}250 \text{ ms}.$$

The experimental results show an evident decrease of the computational time, for $t = 2, 3, 4$. However, these values are slightly higher than the expected ones. In fact, for $t = 2$, the experimental value is 2,696,437 milliseconds instead of 2,278,250. When the number of the nodes is greater than four, then the computational time grows higher and higher. This trend is due to the inter-process communication that requires more time than the computation itself. The worst case is with eight nodes, reporting a value very far from the expected one and very close to the serial time.

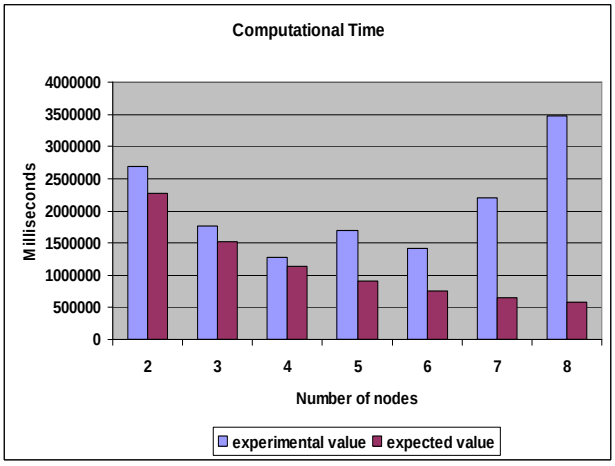

Fig. 4. Comparison of the answer times.

However, in parallel computing, better metrics are the *speedup* and the *efficiency* [18, 19].

The speedup S_t is given by

$$S_t = \frac{T}{T_t},$$

where T is the answer time of the serial program, and T_t is the answer time of the parallel program using t processors. The speedup is represented by a function that shows the gain that is obtained in terms of speed.

A good value of the speedup is:

$$\frac{T}{T_t} = t,$$

represented by a strictly increasing linear function of t.

Figure 5 shows that the speedup increases when $t \leq 4$, while there is a slowdown for $t > 4$, determining a falling of the performance of the system.

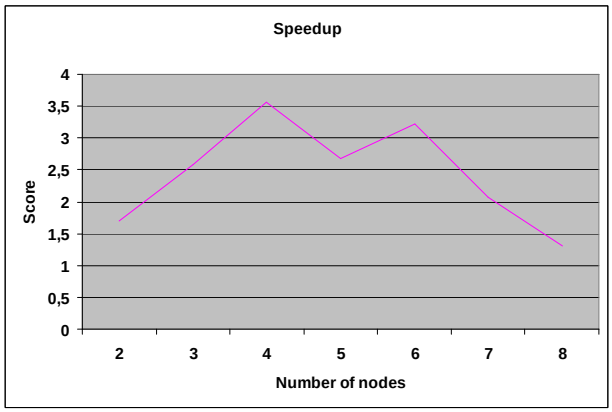

Fig. 5. The Speedup of the parallel algorithm.

On the other hand, the efficiency estimates how good is the computational time, with reference to the number of involved nodes.

The efficiency E_t is given

$$E_t = \frac{S_t}{t}.$$

Figure 6 shows that the efficiency function is always < 1, that is, the parallel algorithm is not able to properly exploit the available nodes. As a consequence, an optimization of the current algorithm PALG is needed in order to obtain a higher level of parallelism. Indeed, experimental results highlight that (a) the computational time is not satisfactory with reference to the number of involved nodes, and (b) the best ratio between the computational time and the number of nodes is obtained for $t = 4$.

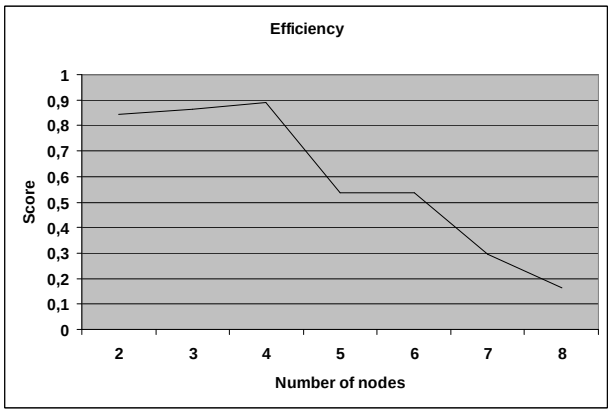

Fig. 6. The Efficiency of the parallel algorithm.

6 Parallel Algorithm Optimization

On the basis of the experimental results, it is possible to state that the best performance is obtained by using four nodes. However, the highest level of parallelism has not yet been reached. In fact, the bottleneck of this algorithm consists of the inter-process communication, whereas there is a high number of data to be transferred and the root node must execute the following loop to get the partial data from the other nodes:

for $q = 1$ *to* $t-1$
    receive(CC', $node_q$)
    …
end for

where t is the number of nodes involved in the parallel computation.

According to this algorithm, the root node is able to gather all the partial results in three steps (*see*, Figure 7):

1. Node 1 sends computed data to root Node 0,
2. Node 2 sends computed data to root Node 0,
3. Node 3 sends computed data to root Node 0.

At the end of these three steps, the root node applies the additive property and obtains the final vector of the CCs.

Notice that in step 1, nodes 2 and 3 are idle, while in step 2 nodes 1 and 3 are idle; at last, in step 3 nodes 1 and 2 are idle. This does not suffice to reach the maximum level of parallelism and led us to optimize the parallel algorithm when executed with four nodes.

The strategy of the optimized version of the parallel algorithm is shown in Figure 8 and allows the root node to gather the partial results in two steps:

1. Node 2 sends computed data to root Node 0 and, at the same time, Node 3 sends computed data to Node 1,
2. Node 1 applies the additive property and sends computed data to root Node 0.

At the end of these two steps, the root node applies the additive property and obtains the final vector of the CCs.

However, this optimization requires the definition of a virtual network topology.

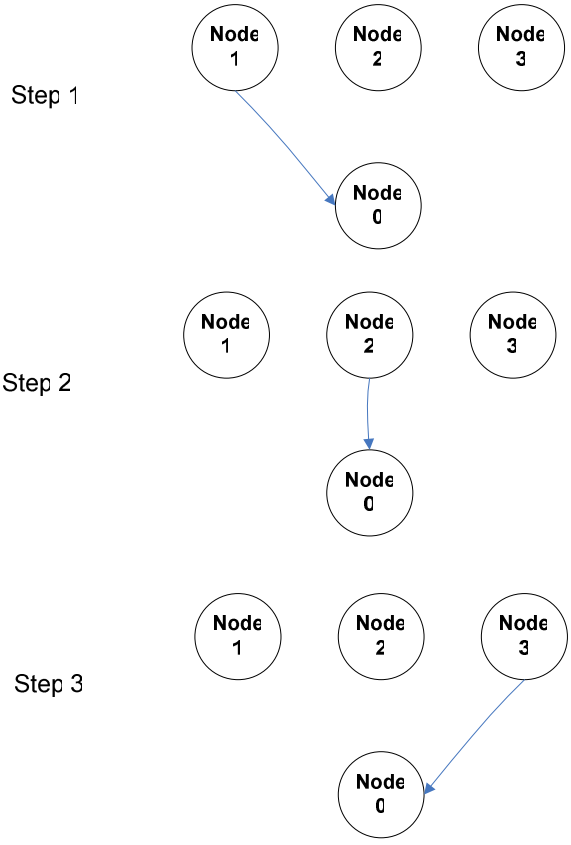

Fig. 7. Parallel algorithm.

6.1 Network topology

Network topology is additional information that can be associated to a parallel architecture and it is just a

mechanism to define different addressing schemes with the processes belonging to a communication group based on MPI [20]. In a few words, a virtual topology describes the ordering of processes according to a geometric shape.

The topology is virtual, that is, it has no impact on the physical layer. The benefit of using a virtual topology consists of naming the processes in the architecture in a way that best fits the communication pattern. As a consequence, writing code results simpler because the processor ranks (*i.e.*, the node identifiers) are based on the topology naming scheme. Moreover, it may also provide hints to the run-time system which allows it to optimize the communication among the nodes.

The most important topology that can be created is the Cartesian topology. Process coordinates in a Cartesian topology begin their numbering at 0, where the numbering strategy is the row-major numbering.

According to this strategy, in an architecture with four nodes, we have that the rank is assigned to each node as follows:
rank 0 → coord(0, 0),
rank 1 → coord(0, 1),
rank 2 → coord(1, 0), and
rank 3 → coord(1, 1).

In this case, processes can be identified by Cartesian coordinates and each process is connected to its neighbours in a virtual grid.

6.2 Optimized parallel algorithm

The following pseudo-code describes the optimized parallel algorithm, in order to be used with four nodes, according to the network topology depicted in Figure 8.

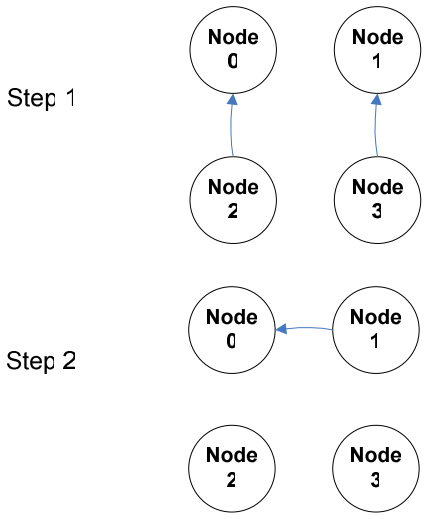

Fig. 8. Optimized parallel algorithm.

Input
n′ = {number of rows} / {number of nodes} = n / t
m = {number of columns}
dg = {approximation degree}
X = {sub-matrix of data}

Output
CC // vector of the Canonical Coefficients, with partial results

Pseudo-code of the parallel algorithm

```
// generation of the coefficients
for d = 0 to dg
for each (d_0, d_1, …, d_{m−1}) such that
    (d_0 + d_1 + … + d_{m−1}) = d
do
for i = 0 to n′−1
pr = Legendre(x′_{i0}, d_0) × Legendre(x′_{i1}, d_1) × … ×
     Legendre(x′_{im−1}, d_{m−1})
end for
CC_z=P
increment index z
end do
end for
// starting inter-process communication

//**************STEP 1
if node identifier = 3 then
        send(CC, node_1)
end if
if node identifier = 2 then
        send(CC, node_0)
end if
if node identifier = 1 then
      receive(CC′, node_3)
        for i = 1 to w
              CC_i = CC_i + CC′_i
        end for
end if
if node identifier = 0 then
      receive(CC′, node_2)
        for i = 1 to w
              CC_i = CC_i + CC′_i
        end for
end if
//**************END STEP 1

//**************STEP 2
if node identifier = 1 then
        send(CC, node_0)
end if
if node identifier = 0 then
      receive(CC′, node_1)
        for i = 1 to w
              CC_i = CC_i + CC′_i
        end for
```

// computation of the mean value
for i = 1 *to w*
$CC_i = CC_i / n$
end for
end if
*//***************END STEP* 2

7 Conclusions

In this paper, we have introduced a novel algorithm for the computation of Canonical Coefficients, that represent the synopsis of the data stored in the Data Warehouse.

These coefficients can be used in order to perform multidimensional analyses of data, obtaining answers affected with a small percentage of error in a lower amount of time.

The C-program has been developed according to a parallel algorithm and it has been tested on a parallel architecture, based on the MPICH library.

The experimental results show that the computational time obtained by the parallel program is much lower than the serial one, when a small number of nodes is involved. In fact, there is no benefit in using many nodes, because of the inter-process communication costs due to the high number of data to be transferred.

However, a deeper analysis of the efficiency of the parallel program highlights that the computational time is always below expectations and that the best performance has been reported with four processors. Therefore, experimental results have suggested the design of an optimized version of the parallel algorithm in order to reach a higher degree of parallelism. This optimized parallel algorithm, expressly designed to be executed with four nodes, has shown a significative improvement of the computational time, by further reducing it of the 5%.

In conclusion, we deem that parallel computing is a good choice for all the batch-applications that do not require an interaction with users and need to perform heavy mathematical computations in background, as always happens in approximation polynomial.

References:

[1] C. dell'Aquila, E. Lefons, and F. Tangorra, Decision portal using approximate query processing, *WSEAS Transactions on Computers*, Vol. 2, No. 2, 2003, pp. 486-492.

[2] K. Naiman, H. Kopackova, S. Simonova, and R. Bilkova, Approaches of Quality Outputs from the Business Systems, *Proceedings of the 5th WSEAS Int. Conf. on Computational Intelligence, Man-Machine Systems and Cybernetics*, Venice, Italy, November 20-22, 2006, pp. 282-285.

[3] A. Datta and H. Thomas, A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases, *Information Systems Research*, Vol. 12, No. 1, 2001, pp. 83-102.

[4] C. dell'Aquila, E. Lefons, and F. Tangorra, Approximate Query Processing in Decision Support System Environment, *WSEAS Transactions on Computers*, Vol. 3, No. 3, 2004, pp. 581-586.

[5] A. Gupta, V. Harinarayan, and D. Quaas, Aggregate-Query Processing in Data Warehousing Environments, *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995, pp.358-369.

[6] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim, Approximate Query-Processing Using Wavelets, *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 2000, pp. 111-122.

[7] P. G. Gibbons and Y. Matias, New Sampling-Based Summary Statistics for Improving Approximate Query Answers, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of data*, Seattle, Washington, United States, 1998, pp. 331-342.

[8] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, Join Synopses for Approximate Query Answering, *ACM SIGMOD Record*, Vol. 28, No. 2, 1999, pp. 275-286.

[9] J. Spiegel and N. Polyzotis, TuG synopses for approximate query answering, *ACM Transactions on Database Systems* (TODS), Vol. 34, No. 1, Art. 3, 2009.

[10] C. dell'Aquila, F. Di Tria, E. Lefons, and F. Tangorra, Data Reduction for Data Analysis. In: C. Cepisca, G. A. Kouzaev, N. E. Mastorakis, *New Aspects on Computing Research*, 2008, pp. 204-210, Athens, WSEAS Press, ISBN/ISSN: 978-960-474-002-2.

[11] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, S. Ramaswamy, and T. Suel, *AQUA: System and Techniques for Approximate Query Answering*, tech. rep., Bell Laboratories, Murray Hill, New Jersey, U.S.A., 1998.

[12] E. Lefons, A. Merico, and F. Tangorra, Analytical profile estimation in database systems, *Information Systems*, Vol. 20, No. 1, 1995, pp. 1-20.

[13] S. Chaudhuri and U. Dayal, An Overview of Data Warehousing and OLAP Technology, *ACM SIGMOD Record*, Vol. 26, No. 1, 1997, pp. 65-74.

[14] MPICH-A Portable Implementation of MPI, www.mcs.anl.gov/research/projects/mpi/mpich1.
[15] The Message Passing Interface standard, www.mcs.anl.gov/research/projects/mpi.
[16] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1999.
[17] C. Reschke, T. Sterling, D. Ridge, D. Savarese, D. Becker, and P. Merkey, A design study of alternative network topologies for the Beowulf parallel workstation, *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, 1996, pp. 626-635.
[18] J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Springer, 1998.
[19] D. L. Eager, J. Zahorjan, and E. D. Lozowska, Speedup Versus Efficiency in Parallel Systems, *IEEE Transactions on Computers*, Vol. 38, No. 3, 1989, pp. 408-423.
[20] J. Larsson Träff, Implementing the MPI process topology mechanism, *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, Maryland, 2002, pp. 1-14.