Parallel Computing for Data Reduction

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

Abstract: - Data Warehouses are databases used in Business Intelligence systems as a data source to develop analytical applications. These applications consist of multidimensional analyses of data and allow decisional makers to improve the business processes of the Information System. Since multidimensional analyses require to aggregate data on several attributes, techniques based on approximate query answering have been introduced in order to reduce the response time. These techniques use, as a data source, a synopsis of the data stored in the Data Warehouse. In this paper, a parallel algorithm for the computation of data synopsis is presented.

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

1 Introduction

A Data Warehouse is a central repository containing large volume of data, obtained by the integration of operational data, coming from heterogeneous source databases. Since a Data Warehouse is used in the decision making process, it must support analytical analyses of data. Moreover, it must allow analyses based on temporal series. For this reason, a Data Warehouse must not only integrate the 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 a Data Warehouse grows in a very fast way, because new records are always added 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 use 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 [1]. In particular, the statistical functions based on the SQL aggregation and grouping operators require a long answering time generally [2, 3].

As the results of statistical computations are used for business strategic choices, decisional makers are not always 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 answering time.

Nowadays, there are several systems supporting approximate query answering, based on different methodologies, such as wavelet [4], sampling [5, 6], or graph-based modelling [7]. In spite of the adopted methodology, all these systems have the following process model in common: (a) calculating the data synopsis and (b) using the calculated data to execute analytical queries. It has been widely demonstrated that these systems are able to produce answers in less time, with an acceptable percentage of error [8].

In particular, the methodology presented here is based on the analytical data profile [9]. According to this methodology, the data synopsis is represented by a set of computed values, the so-called Canonical Coefficients (CC), 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 CC are the coefficients of the approximation polynomial) 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 criteria to evaluate approximate query answering methodologies, we are proposing an extension of our analytic method by designing a parallel algorithm in order to decrease the computational time needed to generate the CC.

Thus, the paper aims to present a parallel algorithm able to compute the CC in a distributed way and to report the experimentation executed according to this novel algorithm. The experimentation 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 CC and introduces the related parallel version of the algorithm. Section 4 shows the parallel architecture used for the experimentation. Section 5 explains the obtained experimental results. Finally, Section 6 reports our conclusions.
2 Approximate Query Answering Systems

An 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 approximate values.

As an example, a decision maker may be interested in knowing who is the best employee of his/her business company in the last year. This information requires to compute 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 is returned quickly.

Here we focus on Approximate Query Answering Systems performing a data reduction. This always happens when using methodologies based on polynomial approximation, sampling, and wavelets. In fact, these systems are able to provide approximate answers using small and pre-computed data synopsis, 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 [10].

The data synopsis is represented by a set of coefficients, used to calculate aggregate functions. Of course, the functions to compute aggregate values are redefined. 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, with no a priori assumptions on the data distribution. At last, these systems can 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, a decision maker can define simple and/or complex business indicators and, then, s/he is allowed to obtain fast and approximate query responses or the exact value against a higher response time.

3 Canonical Coefficients Methodology

This methodology consists of using a polynomial series to approximate the multivariate data distribution function of \( m \) attributes \( X_1, X_2, \ldots, X_m \). The polynomial is the Legendre orthogonal polynomial series and its calculated 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 \( \text{dom}(X_j) = [a_j, b_j] \), for each \( j = 1, 2, \ldots, m \). That is, the domain of attribute \( X_j \) is a numeric (real) interval.

So, we define \( D = [a_1, b_1] \times \cdots \times [a_m, b_m] \).

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 \( y \in \text{dom}(X_j) \) 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:

\[
pdf(x) = g(x) = \frac{1}{2^m} \sum_{i_1=0}^{d} \sum_{i_2=0}^{d} \cdots \sum_{i_m=0}^{d} (2_i + 1) \cdots (2_m + 1) c_{i_1 \ldots i_m} P_{i_1 \ldots i_m}(x'),
\]

where:

- \( x \rightarrow x' = (x_1', x_2', \ldots, x_m') \) is the opportune map from \( x \in D \) to \( x' \in [-1, 1] \);
- \( (i_1, \ldots, i_m) \) is a \( m \)-tuple of natural numbers such that their sum yields \( i \);
- \( P_{i_1 \ldots i_m}(x') = P_{i_1}(x_1') \times \cdots \times P_{i_m}(x_m') \) is the polynomial of degree \( i \);
- \( P_{i_j}(x') \) is the Legendre polynomial of degree \( i_j \);
- \( c_{i_1 \ldots i_m} = \frac{1}{n} \sum P_{i_1 \ldots i_m}(x') \) is the mean value of \( P_{i_1 \ldots i_m}(x') \) on the \( n \) tuples of \( R \).

Therefore, \( g(x) \) is the orthogonal polynomial approximation to \( pdf(x) \) up to degree \( d \) and the coefficients
{c_{i},..., c_{m} \mid i_{1}+...+i_{m} = i, i = 0, ..., 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.

### 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 \( \text{dom}(X_{j}) \) be the domain of the \( j \)-th column. Then, \( \text{Min}_{j} \) and \( \text{Max}_{j} \) denotes respectively the minimum and the maximum of \( \text{Dom}(X_{j}) \). Let \( x'_{ij} \) be the normalization of \( x_{ij} \) in the interval \([-1, 1]\), such that:

\[
\text{x}_{ij} \in [\text{Min}_{j}, \text{Max}_{j}] \Rightarrow x'_{ij} \in [-1,1].
\]

Finally, let \( \text{Legendre}(x'_{ij}, d) \) 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 to generate the CCs.

**Input**
- \( n = \{\text{number of rows}\} \)
- \( m = \{\text{number of columns}\} \)
- \( d_{g} = \{\text{approximation degree}\} \)
- \( M = \{\text{matrix of numeric data}\} \)

**Output**
- \( CC \) /vector of the Canonical Coefficients

**Pseudo-code of the algorithm**

for \( d = 0 \) to \( d_{g} \)

for each \( (d_{0}, d_{1}, ..., d_{m}) \) such that

\[
(d_{0}+d_{1}+...+d_{m}) = d
\]

do

for \( i = 0 \) to \( n-1 \)

\[
pr = \text{Legendre}(x'_{i0}, d_{0}) \times \text{Legendre}(x'_{i1}, d_{1}) \times ... \times \text{Legendre}(x'_{im}, d_{m})
\]

end for

\( CC_{z} = \text{average}(pr) \) //increment index \( z \)

end do

where \( z = 1 \) to \( \frac{d_{g}+m}{d_{g}} \).

**Example 1.** Let \( R(X_{1}, X_{2}) \) be the schema of a relation whose instance is shown in Table 1. For simplicity, let \( d = 2 \) the maximum degree of approximation, \( m = 2 \) the number of fields, and \( n = 4 \) the number of rows.

<table>
<thead>
<tr>
<th>( R )</th>
<th>( X_{1} )</th>
<th>( X_{2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1.**

In this example, \( \text{Min}_{1} = 1 \), \( \text{Max}_{1} = 7 \), \( \text{Min}_{2} = 3 \), \( \text{Max}_{2} = 11 \). Thus, we can normalize each value of the relation \( R \) in the interval \([-1, 1]\). The normalized values are shown in Table 2.

<table>
<thead>
<tr>
<th>( R' )</th>
<th>( X'_{1} )</th>
<th>( X'_{2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>-0.33</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-0.66</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.5</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.**

At this point, the algorithm works in the following manner:

For \( d = 0 \),

\[
(\text{Legendre}(-1,0) \times \text{Legendre}(-1,0) + \text{Legendre}(-0.33,0) \times \text{Legendre}(-1,0) + \text{Legendre}(-0.66,0) \times \text{Legendre}(0.5,0)) / 4 = CC_{1}\]

For \( d = 1 \),

\[
(\text{Legendre}(-1,1) \times \text{Legendre}(-1,0) + \text{Legendre}(-0.33,1) \times \text{Legendre}(1,0) + \text{Legendre}(-0.66,1) \times \text{Legendre}(0.5,0) + \text{Legendre}(1,1) \times \text{Legendre}(0.5,0)) / 4 = CC_{2}\]

For \( d = 2 \),

\[
(\text{Legendre}(-1,2) \times \text{Legendre}(-1,0) + \text{Legendre}(-0.33,2) \times \text{Legendre}(1,0) + \text{Legendre}(-0.66,2) \times \text{Legendre}(0.5,0) + \text{Legendre}(1,2) \times \text{Legendre}(0.5,0)) / 4 = CC_{3}\]

**3.2 Parallel Algorithm**

This Sub-section introduces the parallel version of the algorithm 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 CC [9].

This property states that:
\[
CC_i = \frac{\sum_{j=1}^{t} N_j \times CC'_{ij}}{\sum_{j=1}^{t} N_j},
\]

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 \(CC\). 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 not, 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 numbers of generated \(CC\) (i.e., the cardinality of the vector of the \(CC\)) 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.

**Input**
- \(t\) = \{number of nodes\}
- \(n' = \text{number of rows} / \text{number of nodes} = n / t\)
- \(m\) = \{number of columns\}
- \(dg\) = \{approximation degree\}
- \(M\) = \{sub-matrix of data\}
- \(w\) = \{cardinality of each vector\}

**Output**
- \(CC\) // vector of the Canonical Coefficients, with partial results

**Pseudo-code of the parallel algorithm**

//generation of the coefficients
for \(d = 0\) to \(dg\)
for each \((d_0, d_1, \ldots, d_m)\) such that 
\(d_0 + d_1 + \ldots + d_m = d\)
do
for \(i = 0\) to \(n'-1\)
\(pr = \text{Legendre}(x'_{i0}, d_0) \times \text{Legendre}(x'_{i1}, d_1) \times \ldots \times \text{Legendre}(x'_{im-1}, d_m-1)\)
end for
\(CC_i = pr //\text{increment index } z\)
end do
end for

//starting inter-process communication
if node identifier > 0 then
  // it is not the root node
  send(CC, node0)
else

//it is the root node
for \(q = 1\) to \(t-1\)
  receive(CC', node0)
  for \(i = 1\) to \(w\)
    \(CC_i = CC_i + CC'_{ij}\)
  end for
end for
for \(i = 1\) to \(w\)
  \(CC_i = CC_i / n\)
end for
end if

The limit of this algorithm 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.

### 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). The root node is represented by a Database Server, that 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 communications between a client and the Database Server are 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: “\textit{select ... from ... limit } h, k;\textit{’};”, 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 “\textit{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 [11], an open source implementation of MPI [12, 13].

![Figure 2. Layers of the parallel architecture.](image-url)
Therefore, each node of this 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 by 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 communications among the nodes involved in the computation. On the top, there is the parallel program. 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.

5 Experimental results

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

\[
\binom{27+3}{27} = 4060,
\]

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 3 shows the experimental values obtained in the computation of the CC, in comparison with the expected ones. For the computation of the CC, the expected time is given by:

\[
t = \frac{4,556,500}{t}, \quad \text{for } t = 2, \ldots, 8,
\]

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.

However, in parallel computing, better metrics are the speedup and the efficiency [14, 15].

The speedup \(S_t\) is given by the following formula:

\[
S_t = \frac{T}{T_t},
\]

where \(T\) is the answering time of the serial program, and \(T_t\) is the answering 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:

\[
T = T_t = t,\]

represented by a strictly increasing linear function of \(t\).

![Computational Time vs Number of nodes](image)

Figure 3. Comparison of the answering times.

Figure 4 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. On the other hand, the efficiency estimates how good is the computational time, in reference to the number of involved nodes. The efficiency \(E_t\) is given by the following formula:

\[
E_t = \frac{S_t}{t}.
\]

Figure 5 shows that the function of the efficiency 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 is needed in order to obtain a higher level of parallelism.

Indeed, experimental results highlight that: (a) the computational time is not adequate compared to the involved number of nodes, and (b) the best ratio between the computational time and the number of nodes is for \(t = 4\).

6 Conclusion

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.

This 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 are 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. Therefore, experimental results suggest that the maximum degree of parallelism has not yet been reached and that the parallel algorithm has to be optimized in order to be executed with four nodes. This issue will be addressed in future works.

References