Web-oriented business intelligence solution based on Associative Query Logic

P. Sendín-Raña¹, E. Rodríguez-Fernández¹, F. J. González-Castaño¹,², *, †, E. Costa-Montenegro¹, P. S. Rodríguez-Hernández¹, J. M. Pousada-Carballo¹, J. C. Burguillo-Rial¹

¹Departamento de Ingeniería Telemática, Universidad de Vigo, Vigo, Spain
²Galician Research & Development Center in Advanced Telecommunications, ETSI Telecommunicacion, Campus, 36310 Vigo, Spain

SUMMARY

In this paper we present our experience in the development of a web-based business intelligence tool according to the Associative Query Logic paradigm, which can represent large amounts of data in a way that allows extremely fast queries. It has been developed as an open-source, multi-platform software, relying on data compression techniques for the storage of large amounts of data in the main memory. The performance of our solution in terms of compression, load time and response time is close to that of the commercial tool of reference, QlikView. Moreover, we provide solutions to some open problems in QlikView published description, which may be beneficial to assist in the development of other open or proprietary tools. Copyright © 2010 John Wiley & Sons, Ltd.

Received 7 March 2010; Accepted 7 April 2010

KEY WORDS: business intelligence; Associative Query Logic; data clouds

1. INTRODUCTION

Business Intelligence (BI) refers to a series of technologies used to extract knowledge from businesses and enterprise data [1]. The corresponding tools and applications assist in decision making based on information stored in data warehouses. Basically, BI applications allow users to integrate, store, model and manage relevant data and, from these data records, create, format and show computed results that are meaningful in terms of business knowledge.

Despite the fact that BI applications have experienced an astonishing evolution in the recent years, expert users can identify diverse drawbacks in current tools: they are difficult to deploy and maintain, they take long periods of time to be up and running and they require highly complex interfaces to present results. To sum up, they require end users to have technological expertise.

In this paper we present insights into a software application that can represent large amounts of data in a way that allows extremely fast queries. This tool is based on the Associative Query Logic (AQL) and Data Cloud paradigms.

Our solution has three premises: (a) its software must be open source, (b) it has to be web-oriented and portable, and (c) its performance should be comparable to that of a commercial tool of reference. The result is an efficient management tool that is able to extract data stored in large...
databases, codify them as a compressed dataset in the main memory of a desktop computer and retrieve them using queries in subjectively negligible response time. Once data are transformed, they can be displayed through an eye-catching, simple, yet powerful web interface. In this paper we describe our contributions from a practical perspective. These contributions consist of improvements in the public description of the QlikTech AQL algorithm [2]. As a consequence, the performance of our web-based solution based on interpreted code is similar to that of QlikView, the reference commercial tool by QlikTech International AB. This tool has been highly praised and has a significant international market share nowadays.

This paper is organized as follows: in Section 2 we describe some relevant approaches to BI as the background of our work. In Section 3 we describe the main technologies involved, including AQL and data clouds. Section 4 presents our contributions, leading to the comparative performance evaluation in Section 5. Finally, Section 6 concludes the paper.

2. BACKGROUND

BI, a branch of Decision Supporting Systems (DSS), encompasses data mining software tools and techniques that extract knowledge from enterprise management data. Following Davenport and Prusack [3], we distinguish between information, as a set of processed data with significance and utility, and knowledge, as a mix between information, experience and expertise. Knowledge provides a better understanding of the current state of a company and a more realistic approach to future situations, which results in better and more accurate decision making.

BI software tools rely on ETL (Extract, Transform and Load) techniques that process data from different sources and store them in data warehouses, whose structure allows information to be presented to the end user via graphical interfaces and reports.

There are multiple approaches in the universe of BI systems, ranging from simple querying and reporting systems to complex business forecasting systems. Reference [4] provides a deep and insightful account of the evolution of DSS, from the beginnings of model-driven DSS in the 1960s to On-Line Analytic Processing (OLAP) and Data Warehousing in the 1990s.

OLAP has dominated the BI industry in the recent years. OLAP [5,6] is a formal data mining approach for multidimensional data frameworks. It is an advance over On-Line Transaction Processing (OLTP) of relational databases, given its quick responses to analytical queries. The white paper by Codd et al. [7] stated the well-known 12 rules describing desirable OLAP features. A new definition, which is widely accepted nowadays, was enunciated by Pendse [8], who coined the concept of Fast Analysis of Shared Multidimensional Information (FASMI). According to this definition, an OLAP system satisfies:

- **Fastness**: To deliver most responses to end users in 5 s (1 s for the simplest analyses, 20 s for the more complex ones).
- **Analysis capability**: The system must transparently support any business logic and statistical analysis.
- **Resource sharing**, providing security and locking concurrent updates at appropriate layers.
- **Multidimensionality**: The system must offer a multidimensional view of data. Multidimensionality is defined by multiple hierarchies and levels, in addition to multiple dimensions.
- **Information generation**, regarding both data and related structural information (axis, level, members, etc.).

OLAP systems rely on data warehouses [9] to manage the information, keeping it independent from the operational database in order to optimize query throughput and response times [10]. They support an intuitive approach to large data sets and provide ‘slice and dice’ navigation operations (selection and projection), roll up (progressive reduction in level of detail), drill down (progressive increase in level of detail), and pivot (multidimensional axis redefinition and data view modification).
There exist different approaches to OLAP analytical processing depending on the type of data warehouse:

- **Relational OLAP (ROLAP):** The records are stored in standard relational tables. The combination of a relational database and an OLAP engine is scalable, but the queries are slow due to the complexity of the preprocessing technique. Another disadvantage is the low level of end-user usability.

- **Multidimensional OLAP (MOLAP)** is an array-based model that provides immediate indexing of information for cube queries. Data are stored in a multidimensional database that holds pre-processed data to ensure short response times. The main problem with this approach is its low scalability [11], since it requires vast amounts of storage and may lead to combinatorial explosion.

- **Hybrid OLAP (HOLAP)** seeks to combine the advantages of MOLAP and ROLAP architectures, using an array-based model for summary-type information, and a relational database for detailed information. HOLAP systems balance scalability and speed, but in general their analysis capabilities are limited.

ROLAP is a common developer’s choice given the poor scalability of MOLAP, the disadvantages of HOLAP in practice, and its good compatibility with legacy servers. ROLAP queries rely on summarized data (using aggregate or materialized views in the relational database) for their response times to be practical [12–15]. The challenges of the resulting materialized views are:

- How to identify the aggregate views;
- How to answer queries on materialized views;
- How to keep data consistent while updating materialized views.

However, the design of summarized tables is complex, their maintenance is cumbersome, and it is necessary to keep them consistent when storing or updating information in the data warehouse [16]. In [17], we proposed loading summarized information views in cache memory to avoid consistency problems derived from duplicate data in the underlying relational database.

In ROLAP servers, each multidimensional query is parsed into several sentences for the underlying relational layer. Conversely, there are prototypes that parse relational queries into multidimensional sentences and obtain information from the resulting multidimensional views [18].

OLAP tool interoperability is of great importance. Some proposals are MDAPI [19], OLE DB for OLAP [20], JOLAP [21], and others [22], but none of them has become a commonly accepted standard. Only XML for analysis (XML/A) [23, 24] can be considered a de facto standard.

Currently there exist other research proposals related to OLAP, such as a Grid MOLAP [25]. Thalhammer and Schrefl [26] proposed a new DSS category that automates decision making and extends conventional data warehouses with analysis rules. However, the common approach is a conventional schema combining a passive data warehouse and OLAP tools [27].

The limitations of OLAP systems, due to their inherent complexity, result in long deployment times and long periods of adaptation for the end users to feel comfortable with them. This has given rise to research into new ways of dealing with BI focusing on the user point of view as the paradigm on which this work is based: AQL and data clouds.

### 3. AQL AND DATA CLOUDS

The solution by QlikTech, QlikView [2], consists of linking all stored data through logical associations. This data collection, which is called data cloud, allows users to retrieve information really fast. The associations established among the diverse data structures of the data cloud configure an AQL.
3.1. Advantages and disadvantages of this approach

From a technological point of view, the approach based on AQL and data clouds has the following advantages:

- It allows the creation of flexible and powerful queries, empowering the user to fulfill his management-oriented goals.
- It allows really fast queries. This is because data clouds and AQL allow the retrieval of desired data across several data tables stored in the main memory. With this method, the data search time is linearly dependent on the data size.
- It minimizes main memory usage. First, this method does not impose the data overhead that indexes or summary tables used in other systems require. Second, the data from the secondary memory are compressed and codified into the main memory: each data value is stored only once, and its associated binary-coded value is employed in the implementation whenever needed.
- It allows the integration of data from several sources: different databases, flat files, csv files, etc. These data can be merged in a common data cloud.

The apparent disadvantage of this approach is its lack of persistence: if the source data change during the execution, this change is not reflected in the main memory data. However, this is not a disadvantage in BI scenarios where a 24-h delay in considering newly acquired data is not a problem. Load from source data to main memory can be handled with batch processing and the time it takes is reasonable compared to other solutions.

3.2. Looking inside AQL and data clouds

The two main reasons for the success of QlikView are its simple usage and fast query responses. As the first is related to product implementation, we will focus on the second reason. The response times are low thanks to the ability of managing coded data equivalent to gigabytes of source data in the main memory. Queries are performed on main memory structures instead of on hard disk pages, which significantly decreases the time required to locate and present information. However, it is necessary to work around some problems for this approach to be feasible. First, a good design of data structures is necessary to allow powerful and flexible queries such as those in OLAP systems or conventional RDBMSs (Relational Data Base Management Systems). Second, it is necessary to implement algorithms to compress gigabytes of data (in secondary disks) into megabytes (in main memory). The solutions to both problems must be mutually compatible.

3.2.1. Data clouds. We focus on RDBMS data sources (as in ROLAP systems). These sources follow a table structure. Data element types are called columns, and several tables are linked by some mutual relationships. The goal of a data cloud is to replicate this data structure efficiently in the main memory. Consider Tables I and II, which we will assume have been loaded into the main memory. The tables have multiple columns identified by names. Their corresponding fields in the records are filled with data extracted from the input source.

The relationships between the tables in the main memory are defined by data columns with the same identifier. These relationships represent a logical association between tables and, ultimately, they simplify searching and filtering. Two tables, each with a column with the same identifier, share structures with information about their status in the system. Later in this paper these structures will be described in detail.

| Table I. Structure X, associated with structure Y by ‘Birthplace’. |
|-----------------|-----------------|-----------------|-----------------|
| ID | Name  | Age | Birthplace  |
| 1  | John  | 35  | Paris        |
| 2  | Joan  | 36  | Rome         |
| 3  | John  | 27  | Rome         |
To reduce the size of data structures, the compression mechanism transforms source data values into binary-coded values. Each unique data value from the data source is converted into a binary-coded value, as shown in Tables III and IV.

The data-loading process involves reading data from a source table, creating the associated internal structure in the main memory and codifying the data into binary values. Once all data records have been read, a new structure—the internal table—is created in the main memory, with as many columns as the number of selected columns in the input data table. When the internal table structure is complete, data records are transferred to the main memory. This process consists of reading data values from data source fields and checking whether they appear for the first time or they are repeated. The check is performed against an association table where original data values and binary-coded values are related. If the read data value is already stored in the association table, its previously assigned binary code will be copied inside the corresponding field of the internal table. If the read data value is not found in the association table, a new binary code is created and inserted in the internal table and the association table. There is a different association table for each column in the internal table.

The algorithm must filter data quickly. The data searching process requires three new vectors in addition to the aforementioned tables:

- Status vector
- Selection vector
- Frequency vector.

Each data type has its own associated instances of those vectors. Status vectors and selection vectors are boolean vectors with as many positions as different values in their respective association tables. Each position of these vectors is bound to the corresponding binary value. Thus, position zero in the status vector of a data element type is bound to the binary code zero of the association table of the data element type. Frequency vectors have the same number of components, but their content is numerical.

Selection vectors represent the selection that is made in the graphical interface at the client side. The fields that a user selects will be identified by a true value, whereas the remaining fields of that specific column will be false. Status vectors represent the current state of the system: from the collection of status vectors it is possible to identify which fields from different columns are related to the ones the user selects through the graphical interface. Status vectors are modified through the
search process. As in the case of selection vectors, there is a status vector for each data element type and for each status vector there are direct relationships between its positions and the number of different data values in the corresponding association table. Figure 1 shows a set of selection and status vectors and their representation in the user interface when the user selects ‘John’ and ‘36’ in the scenario defined by Tables I and II. Frequency vectors allow internal information to be maintained across the searching process.

3.2.2. Filtering process. The filtering process starts with a user selection in the graphical interface. The selection is defined in the set of selection vectors by assigning value one (true) to the selected fields and value zero (false) to non-selected ones. If the selection is the first one or if it implies a status restart (because it is less restrictive than a previous one), all the components in the status vectors are set to one, and all the frequency vectors are cleared.

In the next step, the selection vectors that have changed are copied into the corresponding status vectors. Then, all the data records from the internal table are checked. For each field in the data records, the algorithm checks if the binary-coded value is activated within its corresponding status vector. If this is true for all the fields of the record, the algorithm determines that there is a match, and adds one to the corresponding position of the associated frequency vector. Once all the records of the data table have been checked, the status vectors are modified according to the frequency vectors: if a position of a given frequency vector has a non-zero value, the associated position of the corresponding status vector keeps a true value. Otherwise, the position is cleared. After these modifications, the frequency vectors are cleared.

This process continues for the remaining internal data tables of the system. A status vector that is modified in a given table affects other tables that share that vector. Once all tables have been checked, the algorithm checks if at least one status vector has changed from its initial values. If this has occurred, the process starts again from the first internal data table. Otherwise, the algorithm concludes and the set of status vectors represent the current state of the system.

Figure 2 shows a representation of the algorithm. A detailed description can be found in [28, 29].

4. WEB-ORIENTED AQL BI SOLUTION

Given the many benefits of open-source software, there exist successful open-source BI products on the market, such as Pentaho [30] or JasperSoft [31]. However, these approaches are OLAP oriented. At the time this paper was written, there were no solutions based on AQL or data cloud models.

4.1. Service-oriented architecture

We wish to provide an environment for publishing and invoking BI services across the Web, as a fully interactive and highly visual web-based application using well-known and widely used
WEB-ORIENTED BI SOLUTION BASED ON AQL

Figure 2. Filtering flow diagram.

technologies. For this reason, our web services architecture follows an MVC (Model-View-Controller) pattern [32], more specifically Java Struts [33], an open-source web application framework. We employ SOAP (Simple Object Access Protocol) [34] for exchanging structured information.

The interactive web-based application at the client side is based on AJAX (Asynchronous JavaScript and XML) [35]. AJAX is an important web development model for web browser-based applications. The client also relies on JSP (Java Server Pages) to issue initial requests to the Java server. SOAP response messages are then transformed and presented to the client browser. For content presentation, CSS (Cascading Style Sheets) are used.

SOAP request messages are defined with the following XML schema:

```xml
<element name="getDataRequest">
  <complexType>
    <sequence>
      <element name="idUser" type="int"/>
    </complexType>
  </element>
  <complexType>
    <complexContent>
      <complexType>
        <sequence>
          <element name="idUser" type="int"/>
        </complexType>
      </complexContent>
      <complexType>
        <attribute name="case" type="string" use="required"/>
      </complexType>
    </sequence>
    <element minOccurs="0" name="filter" type="string" use="required"/>
    <element minOccurs="0" name="object" type="string" use="required"/>
  </complexType>
</element>
```

SOAP response messages are defined with the following XML schema:

```
<element name="getDataResponse">
  <complexType>
    <sequence>
      <element minOccurs="0" name="filters">
        <complexType>
          <attribute name="case" type="string" use="required"/>
        </complexType>
      </element>
      <element minOccurs="0" name="object">
        <complexType>
          <sequence>
            <element name="id" type="string"/>
            <element minOccurs="0" name="text" type="string"/>
            <element minOccurs="0" name="idMatrix" type="string"/>
            <element maxOccurs="unbounded" minOccurs="0" name="classification">
              <complexType>
                <simpleContent>
                  <extension base="string">
                    <attribute name="field" type="string" use="optional"/>
                  </extension>
                </simpleContent>
              </complexType>
            </element>
            <element minOccurs="0" name="expression" type="string"/>
          </sequence>
          <attribute name="new" type="boolean" use="required"/>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
```

The server side has been developed using Java technology. Although an interpreted language such as Java is not optimal regarding response time or memory consumption, it offered several interesting advantages in our case. As previously mentioned, a major goal of our research project was to provide a fully interactive, highly visual, web-based application, with features such as privacy, persistence, and security. In this context, some of the advantages of Java technology are:

- Security and robustness through its powerful exception handling, memory allocation, high-quality error debugging and automatic garbage collection mechanisms.
- Platform independence and code portability.
- Support by a broad, highly active community, and large number of available libraries. In addition, it allows easy integration with multiple previously developed Java applications.
- A Java BI application can be fast enough: we are not interested in beating a commercial product like QlikView, but in achieving comparable performance with an open-source, portable, web-based solution.

4.2. Algorithm improvements

During our project, we detected some gaps in the theoretical description of the data cloud/AQL algorithm as well as in the commercial products derived from it.

4.2.1. Efficient data load and storage. QlikView loads data columns and renames them with aliases. According to QlikView published description [28], the same column may be renamed several times with different aliases, and the original data column is loaded as many times as different aliases identify that column.

Columns are stored internally with different identifiers, their aliases. By doing so, multiply loaded columns are completely independent from the application point of view and applying a filter to one does not affect the others. Nonetheless, loading duplicate data wastes time and memory space in the data loading process (which may involve millions of data records).

Our implementation overcomes this drawback. The idea consists of identifying the column names that are read from the input source and the aliases that are used to rename those columns, and the application of different filters. Therefore, we define a vector with the names of the data element types (naming vector) and another vector with their respective aliases (alias vector). It is necessary to establish a relationship between these two vectors to identify the aliases that correspond to a name. Given that the naming vector represents the column structure of the binary table, and that it shares the same column identifiers, we decided to create a third structure whose records hold the names of the columns and the aliases of the columns. Binary data columns are linked to association tables, and for that reason we also defined a connection between the aliases and the corresponding association table.

If we wish to apply independent filters to data coming from the same input data column, we need to assign different selection, status and frequency vectors to each alias. In doing so, the status of a coded data column in the memory can differ from the status of another coded data column in the memory resulting from the same input data column. Essentially, this is possible because they have different status vectors.

Figure 3 illustrates this methodology, where a naming vector, an alias vector with its associated records, and a binary data table with its association tables are represented. This simple solution allows us to decrease data-loading time and main memory space because the binary data columns are loaded only once.

Figure 3. Design of the methodology to reduce load time and memory space.
4.2.2. Establishing an order to check the tables. In the data filtering process we found that the data cloud/AQL methodology does not always perform as described in [28, 29, 36]. In fact, the description only works in a reduced number of cases. We will explain this statement with an example.

Table V shows two tables with two columns each, which are linked by the id data column. In this simple scenario we can request the name and the age of a user as well as his id number and the relations between them. By picking one of the values of a column we can obtain the related information from the other two columns. If, for example, we want to know the name and the age of the user with id #1, after performing the filtering algorithm, we determine that his name is ‘Mark’ and his age is ‘27’. After filtering, the status vectors for id, name and age are \(\langle 1, 0 \rangle\), \(\langle 1, 0 \rangle\), and \(\langle 1 \rangle\), respectively. This is coherent with the expected result.

If we pick id #2 (id selection vector \(\langle 0, 1 \rangle\)) the expected result will be the name ‘John’ without age information (id status vector \(\langle 0, 1 \rangle\), name status vector \(\langle 0, 1 \rangle\), and age status vector \(\langle 0 \rangle\)). However, the set of status vectors that are obtained after applying the published algorithm description differs from the expected result. Let us analyze the evolution of the status vector and the frequency vector through the filtering process.

First, the id selection vector is copied into its status vector. Next, the algorithm checks the first table, updating the frequency vectors. There is a match for the user with id #2, hence it is necessary to update the status vectors of id and name once the first table is completely checked. The new status vectors for id and name are \(\langle 0, 1 \rangle\) and \(\langle 0, 1 \rangle\), respectively. Turning to the second table we must use the id and age status vectors. The id status vector has changed to \(\langle 0, 1 \rangle\) and the age status vector stays as \(\langle 1 \rangle\). After checking the second table, the id status vector becomes \(\langle 0, 0 \rangle\), and the age status vector is \(\langle 0 \rangle\), since no match was found in any of the data records of the second table. At this point, all tables have been checked and at least one status vector has changed, so the algorithm must be executed again using the new status vectors.

We restart the table checks beginning with the first table. The values of status vectors id, name, and age are \(\langle 0, 0 \rangle\), \(\langle 0, 1 \rangle\), and \(\langle 0 \rangle\). After checking all the data records in the first table, the name status vector changes to \(\langle 0, 0 \rangle\). Repeating the process in the second table, the id and age status vectors do not change, thus they are \(\langle 0, 0 \rangle\) and \(\langle 0 \rangle\), respectively. Again, once the algorithm has checked all the data tables, at least one status vector has changed, hence a third check of the two tables is necessary. After this last check, the status vectors for id, name and age stay the same, and the algorithm finishes, reporting that there are no responses for id #2, which, obviously, is not the expected result.

On investigating the algorithm we found that the problem is due to the second table, which did not have a record field with id #2. After checking this table, the id status vector changed, losing its representation for id #2. When we re-checked the first table the new status vector caused an error.

We can, therefore, conclude that when the published algorithm is used, if one of the tables of the system does not contain all the different data values for a given column, and a selection is made in that column, the actual result differs from the expected one. This is independent of table checking order.

This flaw can be overcome by introducing some changes in the algorithm. The idea is to establish a particular downfall order to check the tables and keep the status vector for the connecting column unchanged. The algorithm must create a tree structure with a root node and several leaf nodes that can be directly reached from the root or through multiple connecting nodes. The first step consists of identifying the root node or starting table, which contains the data column of the selected value. If the column of the selected value belongs to several tables we can choose any of them.
After determining the starting table, the next step consists of identifying the relationships between the starting table and the other tables of the structure. As we have seen, two tables are related if they have a data column with the same identifier. So, the algorithm will choose a table with a data column that also belongs to the starting table as the next table to check. If there are several tables with this first level of relationship with the starting table, any of them can be selected without loss of functionality.

At this point, the algorithm must determine if the current table node is a leaf or a connecting node. Leaf tables only have one data column in common with another table, and this column is the link between the current node and the upper-level node. On the other hand, a connecting node has several links, that is, two or more data columns shared with other tables. It is important to realize that a connecting node may have several links with the bottom level, but only one link with the upper level. Otherwise, there would be a loop in the structure.

If the table is not a leaf node, we follow the structure until we find one, instead of first selecting all the nodes at the same level and then continuing down to the next level, but the algorithm behaves correctly in both cases. Once a leaf node is reached, the algorithm climbs along the path to that node until it finds the first node that has not been included in the searching order yet. Then the algorithm goes down to another leaf node and keeps repeating this process until all the tables have been included in the searching order.

On completion of the process, the correct order to traverse the structure of tables is known (we follow the preordered traversing sequence), as are the data columns that define the connections between them. This is an important issue, because while a table is being checked, all the status vectors may change except those that define the correct traversing order: the status vector of the selected value in the starting table and the status vector of the data column connecting all the other tables with the table at the level immediately above. These status vectors must remain unchanged while the corresponding table is being checked, but they can be modified if they are contained in another table.

We will explain how the new algorithm works with an example. There are five parts of Table VI, three related by field name, and one related to the other two by a different field, studies. We can see that the data column of the selected value is contained in three different table parts. Thus a valid order to traverse the table parts could be (a)–(b)–(d)–(e)–(c). The user selection is in dark gray and the filtering results are in light gray.

After the starting Table VI(a) has been checked, the status vectors for the name and age data columns are \((1, 0, 0)\) and \((1, 0)\), respectively. The age status vector changes but the name status vector remains the same. Then we check Table VI(b). In this, the data column name is the connecting part that should remain unchanged, but since the current status vector is the user selection vector, it would not change anyway. However, the studies status vectors change to \((1, 0, 0, 0)\). Without any further complication we traverse Tables VI(d) and (e), in that order, and find that the status vectors for semesters and courses are \((1, 0)\) and \((1, 1, 0, 0)\), respectively.

The last part that is traversed is Table VI(c). After this is traversed, the name status vector should change to \((0, 0, 0)\), because ‘John’ is not listed in that part and the other positions of the vector are already set to zero. However, this is the connecting data column, so it should remain unchanged and the name status vector keeps its value \((1, 0, 0)\). The location status vector changes to \((0, 0)\). Therefore, with our modification of the algorithm we can reach the expected solution. Moreover, this is achieved with a single check of the table. Table VII shows the final status vectors.

Other combinations of filters, regardless of their complexity, can be applied and solved correctly using our modification. Just like in the original algorithm, more restrictive filters can be directly applied from the current set of status vectors, whereas less restrictive filters require a reset restoring the initial value of all the status vectors.

4.3. Using the improved algorithm to extract results

The coded information stored in the main memory must be extracted efficiently to allow the evaluation of mathematical functions on the selected variables. The methodology in [31] defines the
steps required to identify all boundary and connecting tables, to select the starting table to begin the conversion process and to evaluate the mathematical function for each data record. However, the conversion process generates a data structure that is not suitable for all filtering cases. When data filters are not part of the generated structure, the selection vectors for these filters are not considered and the obtained results are not as expected. Unacceptably long delays arise if we implement all these steps every time the user makes a selection.

Our proposal considers data filters in the data structure that is created to support tables or charts in the graphical user interface. First, the application server obtains the expressions (functions) to evaluate, and then the data variables are classified by the different dimensions defined. The next step is to determine the dimension axis and its size, and an order in which to traverse the tables. Once the algorithm has this information and the variables are identified (creating volatile data structures), the route-generation process starts, and a new binary-coded structure is created. The route generation process begins from the tables where the data variables are defined and ends at the dimension tables (defined by the user via the graphical interface), traversing all the connecting tables and collecting only the relevant information (for mathematical or filtering functions).
The binary-coded data for table or chart representation is implemented in an efficient structure to avoid memory overload and allow fast response times. All the possible filters defined on the dashboard can be selected from this new data structure. Figure 4 shows a load script to build the in-memory data structure. A dashboard with three lists (filter operations) and three charts is shown in Figure 5. As there are no user selections, all the aggregated data are presented. In Figure 6, the user selects one product, and the sales of this product are shown by location. Figure 7 illustrates another configuration, with shop selection: product list and state list are affected by the shop filter, as is the graphical sales information by product.

5. NUMERICAL TESTS

We have evaluated our contributions in terms of load time, search time, and amount of main memory used. We take QlikView as a reference to compare performances. Note that our web-oriented application implementation with multi-user support requires extra transmission time between client and server. Conversely, QlikView is a local tool.
The following tests were carried out on a Dell computer (acting as a server) with a 2.3 GHz AMD ATHLON processor with 2 GB DDR2 SDRAM. Dashboards were tested with the more popular browsers:

- Mozilla Firefox
- Internet Explorer 8
- Google Chrome
- Opera

In the first test we compared the times required to load data from relational tables into the main memory. We defined this time as the time from the request to load source data to when the user is informed that the data have been loaded into the main memory. Figure 8 shows the times for different load scripts, involving different numbers of registers loaded. These times also depend on the type of data and the values that are repeated for each column. On average, the size of the source data in a MySQL engine was 550 MB.

In the second test, we measured the memory consumption of the two approaches after the data were loaded into the main memory (Figure 9). Again, we must consider that these results depend on repeated values, type of data loaded and number of registers. Although our proposal has a memory consumption penalty, we consider this increase more than acceptable, taking into account our principal premises: a multiplatform implementation, suitable for any OS, with a
client-server architecture allowing multiuser support. Many users from different locations may be sending queries and receiving results at once, which globally implies a more efficient memory usage. Given that the size of the source data was 550 MB on average, the relative difference between both approaches is low.

Figure 10 shows the memory consumption of the proposed solution during a load process, from the command to load the data structure to the moment at which the user is informed that the data have been loaded into the main memory. The final state (73 MB) corresponds to one of the cases that are depicted in Figure 9. There is a peak in memory consumption due to the intermediate data structures that are used to build the final binary-coded values, but the collector garbage of JVM releases this temporal information quickly, and a low-end machine can handle the temporary peak.

Figure 11 shows the memory consumption when aliases are used in the data loading process. It illustrates the improvement of our proposal, which identifies the column names and the aliases that are used in the data load script. The QlikView implementation stores redundant information,
Figure 10. Memory consumption in the data load process. The horizontal axis represents elapsed time.

Figure 11. Memory consumption comparison when aliases are used in the data loading process.

whereas our proposal only creates new status, selection and frequency vectors. BI interfaces typically allow users to arrange the same information by different filters in different tabular or chart representations. Taking this into account, a shared dashboard (which implies several aliases) based on our implementation is more efficient than QlikView in terms of overall memory consumption.

The response time for chart creation was less than one second at all times in our experiments. Usually this response time includes the time needed to filter one or several selected variables. Figure 12 shows the filter time and Figure 13 shows the times required for chart creation (or tabular representation) in the same scenario of the first and second tests. These response times are adequate from a subjective point of view.

6. CONCLUSIONS

In this paper we have presented a solution for implementing an efficient BI web tool that can be used by end users without technical expertise to perform complex analyses. This is the first implementation of the Data Cloud/AQL algorithm that is open and portable. Its performance in terms of loading time, main memory consumption, and search time are comparable to those of the
commercial tool of reference, QlikView. Besides, we provide two improvements of the original Data Cloud/AQL algorithm, which may be greatly beneficial even for optimized machine-dependent local applications (such as QlikView). Specifically, these improvements are a significant reduction in the memory information required and a redefinition of the search algorithm that avoids some errors in the QlikView published description.

REFERENCES

23. XML/A. Available at: http://www.xmlforanalysis.com [January 2010].
34. SOAP Specifications. Available at: http://www.w3.org/TR/soap [January 2010].