COST-EFFECTIVE DATA ALLOCATION IN DATA WAREHOUSE STRIPING

Raquel Almeida[1], Jorge Vieira[2], Marco Vieira[1], Henrique Madeira[1] and Jorge Bernardino[3]

[1]CISUC, Dept. of Informatics Engineering, Univ. of Coimbra, Coimbra, Portugal
{rrute, mvieira, madeira}@dei.uc.pt
[2]CISUC, Critical Software SA, Coimbra, Portugal
jvieira@criticalsoftware.com
[3]CISUC, ISEC, Coimbra, Portugal
jorge@isec.pt

ABSTRACT

The Data Warehouse Striping (DWS) technique is a data partitioning approach especially designed for distributed data warehousing environments. In DWS the fact tables are distributed by an arbitrary number of low-cost computers and each query is executed in parallel by all the computers, guarantying a nearly optimal speed up and scale up. Data loading in distributed data warehouses is typically a heavy process and brings the need for loading algorithms that conciliate a balanced distribution of data among nodes with an efficient data allocation. These are fundamental aspects to achieve low and uniform response times and, consequently, high performance during the execution of queries. This paper proposes a generic approach for the evaluation of data distribution algorithms and assesses several alternative algorithms in the context of DWS. The experimental results show that the effective loading of the nodes must consider complementary effects, minimizing the number of distinct keys of any large dimension in the fact tables in each node, as well as splitting correlated rows among the nodes.

KEYWORDS

Data distribution, Data striping, Data warehousing, Performance

1. INTRODUCTION

Data warehouses represent nowadays an essential source of strategic information for many companies. In fact, as competition among enterprises increases, the availability of tailored business information that helps decision makers during decision support processes is of utmost importance.

A data warehouse (DW) is an integrated and centralized repository that offers high capabilities for data analysis and manipulation [1]. In a data warehouse the data is organized according to the multidimensional model, which includes facts and dimensions. Facts are numeric or factual data that represent a specific business or process activity and each dimension represents a different perspective for the analysis of the facts. The multidimensional model is typically implemented as one or more star schema made of a large central fact table surrounded by several dimensional tables related to the fact table by foreign keys.

Typical data warehouses are periodically loaded with new data that represents the activity of the business since the last load [2]. This is part of the normal life cycle of data warehouses and includes three key steps (also known as ETL): Extraction, Transformation, and Loading. In

DOI: 10.5121/ijdms.2012.4509

practice, the raw data is extracted from several sources and it is necessary to introduce some transformations to assure data consistency, before loading that data into the DW.

Data warehouses store high volumes of data integrated from several different operational sources. Thus, the data stored in a DW can range from some hundreds of Gigabytes to dozens of Terabytes. Obviously, this scenario raises two important challenges. The first is related to the storage of the data, which requires a large and highly available storage system. The second concerns accessing and processing the data in due time, as the goal is to provide low response times for the queries issued by the users.

In order to properly handle large volumes of data, allowing to perform complex data manipulation operations, enterprises normally use high performance systems to host their data warehouses. The most common choice consists of systems that offer massive parallel processing capabilities [3], [4], as Massive Parallel Processing (MPP) systems or Symmetric MultiProcessing (SMP) systems. Due to the high price of this type of systems, some less expensive alternatives have already been proposed and implemented [5], [6]. One of those alternatives is the Data Warehouse Stripping (DWS) technique [7], [8], a solution already implemented and made available commercially by Critical Software S.A. [9].

In the DWS technique the data of each star schema of a data warehouse is distributed over an arbitrary number of nodes. This way, a major challenge faced by DWS is the distribution of data to the cluster nodes. In fact, DWS brings the need for distribution algorithms that conciliate a balanced distribution of data among nodes with an efficient data allocation. Obviously, efficient data allocation is a major challenge as the goal is to place the data in such way that guarantees low and uniform response times from all cluster nodes and, consequently, high performance during the execution of queries.

This paper extends a preliminary work done by the authors in [10] and proposes a generic methodology to evaluate and compare data distribution algorithms. The approach is based on a set of metrics that characterize the efficiency of the algorithms, considering three key aspects: data distribution time, coefficient of variation of the number of rows placed in each node, and queries response time. Data and queries from the TPC-DS performance benchmark [11] are used to exercise the data distribution algorithms. The methodology includes a set of steps that should be followed during the evaluation of an algorithm, which makes the approach generic and suitable for other data distribution algorithms besides the ones addressed in this paper.

The paper studies three key data distribution algorithms that can be used in DWS clusters: round-robin, random, and hash-based. Concerning the round-robin algorithm, several variants are addressed considering different loading windows, namely: round-robin 1, round-robin 10, round-robin 100, round-robin 1000, and round-robin 10000. To demonstrate the proposed methodology and evaluate the data distribution algorithms a set of experiments was conducted using the PostgreSQL Database Management System (DBMS).

The goal of these experiments is to identify the data distribution algorithm that best fits the DWS needs, as well as trying to establish the most relevant characteristics that can make a data distribution produce the best system performance.

The structure of the paper is as follows. Section 2 presents related work. Section 3 discusses the data distribution problem in the context of DWS. Section 3 presents the methodology for the evaluation of data distribution algorithms. The experimental evaluation is described in Section 4 and Section 5 concludes the paper.

2. RELATED WORK

There is a vast literature on query processing and load balancing in parallel database systems (e.g., [12-14]) and distributed databases (e.g., [15-17]). In [15] it is discussed the potential of parallel processing in the data warehouse loading process and for the maintenance of materialized views. However, this work does not address the use of parallel technology for data warehouse analysis.

Many DBMS vendors claim to support parallel data warehousing to various degrees, including: Oracle 11g [18], IBM/Informix Red Brick [19], and the Microsoft SQL Server [20]. Most of these products, however, do not take advantage of dimensionality of data that exists in a data warehouse and it remains unclear to what extent multidimensional fragmentation is exploited to reduce query work. None of the aforementioned vendors provide sufficient information or even tool support on how to determine an adequate data allocation for star schemas. In our opinion, the effective use of parallel processing in data warehouses can be achieved only if we are able to find innovative techniques for parallel data placement using the underlying properties of data.

One of the first works to propose a parallel physical design for the data warehouse was DATAllegro. This work proposes a data indexing strategy based on vertical partitioning of the star schema to provide efficient data partitioning and parallel resource utilization. The paper presents algorithms that split the data among N parallel processors and perform parallel join operations, but without quantifying potential gains. A multidimensional hierarchical fragmentation and allocation method for star schemas in a parallel data warehouse environment was recently proposed in [21]. This approach called MDHF (MultiDimensional Hierarchical Fragmentation) allows all star queries referencing at least one attribute from any fragmentation dimension to be confined to a subset of the fact table fragments. This approach assumes a shared disk parallel database system that exhibits near linear scalability with respect to the number of disks and processors, but in certain cases (partially filled bitmap indexes) it shows an increase in I/O load and has some administration overhead.

Although we are implementing a centralized data warehouse distributed over a computer network environment, our work is also related to distribute processing in data warehouses. The fact that many data warehouses tend to be extremely large in size [22] and grow quickly means that a scalable architecture is crucial. A truly distributed data warehouse can be achieved by distributing the data across multiple data warehouses in such a way that each individual data warehouse cooperates to provide the user with a single and global view of the data. In spite of the potential advantages of distributed data warehouses, especially when the organization has a clear distributed nature, these systems are always very complex and have a difficult global management [23]. On the other hand, the performance of distributed queries is normally poor, mainly due to load balance problems.

In this context, DWS provides a flexible approach for distribution, inspired in both distributed data warehouse architecture and classical round-robin partitioning techniques. The data is partitioned in such a way that the load is uniformly distributed to all the available computers and, at the same time, the communication requirements between computers are kept to a minimum during the query computation phase [7], [8].

A considerable body of research has been performed for processing and optimizing queries over distributed data (see, e.g. [24-29]). However, this research has focused mainly on distributed join processing rather than distributed computation. The approach we explore in this paper marries the concepts of distributed processing and data placement to provide a fast and reliable relational data warehouse.

3. DATA DISTRIBUTION IN DWS NODES

Data Warehouse Striping allows enterprises to build large data warehouses at low cost. DWS can be built using inexpensive hardware and software (e.g., low cost open source database management systems) and still achieve very high performance. In fact, DWS data partitioning for star schemas balances the workload by all computers in the cluster, supporting parallel query processing as well as load balancing for disks and processors. The experimental results presented in [7] show that a DWS cluster can provide an almost linear speedup and scale up.

In the DWS technique [7], [8] the data of each star schema of a data warehouse is distributed over an arbitrary number of nodes having the same star schema (which is equal to the schema of the equivalent centralized version). The data of the dimension tables is replicated in each node of the cluster (i.e., each dimension has exactly the same rows in all the nodes) and the data of the fact tables is distributed over the fact tables of the several nodes (see **Figure 1**). It is important to emphasize that the replication of dimension tables does not represent a serious overhead because usually the dimensions only represent between 1% and 5% of the space occupied by all database [1]. In the rare cases in which the star schema has a very large dimension it is possible to accommodate that dimension in the DWS cluster by using selective loading techniques [30] or encoding techniques [31].

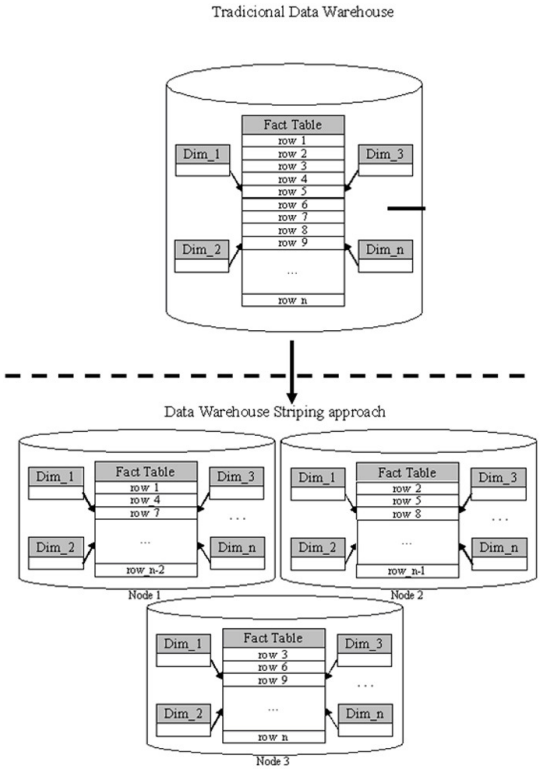

Figure 1. Traditional approach *vs* Data Warehouse Striping Technique

One of the challenges faced by DWS is data loading, as distributing large amounts of data to the cluster nodes must be an efficient process. Original DWS proposal [7] did not address this issue in depth and all performance tests conducted assumed that it would be possible to define and implement an efficient data loading mechanism, which is not always the case. In fact, poorly distributed data may lead some nodes to process more data than others, which may affect the system's response time.

In a DWS cluster OLAP (On-Line Analytical Processing) queries are executed in parallel by all the nodes available and the results are merged by the DWS middleware (i.e., middleware that allows client applications to connect to the DWS system without knowing the cluster implementation details). Thus, if a node of the cluster presents a response time higher than the others, all the system is affected, as the final results can only be obtained when all individual results become available [7]. This means that the overall query execution time is defined by the slowest node, and so the data loading algorithm must ensure that the data is distributed in such a way that the slowest node is as fast as possible.

In a DWS installation, the extraction and transformation steps of the ETL process are similar to the ones performed in typical data warehouses (i.e., DWS does not require any adaptation on these steps). It is in the loading step that the nodes data distribution takes place. Loading the DWS dimensions is a process similar to classical data warehouses; the only difference is that they must be replicated in all nodes available. The key difficulty is that the large fact tables have to be distributed by all nodes.

Loading facts data in DWS is a two steps process. First, all data is prepared in a DWS Data Staging Area (DSA). This DSA has a data schema equal to the DWS nodes, with one exception: fact tables contain one extra column, which will register the destination node of each row. The data in the fact tables is chronologically ordered and the chosen algorithm is executed to determine the destination node of each row. In the second stage, the fact rows are effectively copied to the node assigned. Three key algorithms can be considered for data distribution:

- **Random data distribution**: The destination node of each row is randomly assigned. The expected result of such an algorithm is to have an evenly mixed distribution, with a balanced number of rows in each of the nodes but without any sort of data correlation (i.e. no significant clusters of correlated data are expected in a particular node).

- **Round Robin data distribution**: The rows are processed sequentially and a particular predefined number of rows, called a window, is assigned to the first node. After that, the next window of rows is assigned to the second node, and so on. For this algorithm several window sizes can be considered, for example: 1, 10, 100, 1000 and 10000 rows (window sizes used in our experiments). Considering that the data is chronologically ordered from the start, some effects of using different window sizes are expected. For example, for a round-robin using size 1 window, rows end up chronologically scattered between the nodes, and so particular date frames are bound to appear evenly in each node, being the number of rows in each node the most balanced possible. As the size of the window increases, chronological grouping may become significant, and the unbalance of total number of facts rows between the nodes increases.

- **Hash-based data distribution**: In this algorithm, the destination node is computed by applying a hash function [32] over the value of the key attribute (or set of attributes) of each row. The resulting data distribution is somewhat similar to using a random approach, except that this one is reproducible, meaning that each particular row is always assigned to the same node.

4. EVALUATING DATA DISTRIBUTION ALGORITHMS

Characterizing data distribution algorithms in the context of DWS requires the use of a set of metrics. These metrics should be easy to understand and be derived directly from experimentation. We believe that data distribution algorithms can be effectively characterized using three key metrics:

- **Data distribution time (DT)**: The amount of time (in seconds) a given algorithm requires for distributing a given quantity of data in a cluster with a certain number of nodes. Algorithms should take the minimum time possible for data distribution. This is especially important for periodical data loads that should be very fast in order to make the data available as soon as possible and have a small impact on the data warehouse normal operation.

- **Coefficient of variation of the amount of data stored in each node (CV)**: Characterizes the differences in the amount of fact rows stored in each node. CV is the standard deviation divided by the mean (in percentage) and may be particularly relevant when homogenous nodes are used or the storage space needs to be efficiently used. It is also important to achieve uniform response times from all nodes.[1]

- **Queries response time (QT)**: Characterizes the efficiency of the data distribution in terms of the performance of the system when executing user queries. A good data distribution algorithm should place the data in such way that allows low response times for the queries issued by the users. As query response time is always determined by the slowest node in the DWS cluster, data distribution algorithms should assure well balanced response times at node level. QT represents the sum of the individual response times of a predefined set of queries (in seconds).

To obtain these metrics we need data and a set of queries to explore that data. The data is initially distributed over the cluster nodes using the algorithm being assessed. Afterwards, the queries are executed to obtain response times. In our approach we use the recently proposed TPC Benchmark DS (TPC-DS) [11], as it models a typical decision support system, imitating the activity of a multi-channel (stores, catalogs, and the Internet) retailer, thus adjusting to the type of systems that would be implemented using the DWS technique. The TPC-DS schema is a star schema, consisting of multiple dimension tables and seven fact tables, modeling the sales and sales returns processes of the business considered. A key advantage of using TPC-DS is that is has been the result of an extensive study by the Transaction Processing Performance Council to define a data warehouse model and a set of queries that are representative of real systems from the field. The size of the TPC-DS database (defined in the specification as a scaling factor) should be chosen taking into consideration the number and capabilities of the nodes in the cluster.

Evaluating the effectiveness of a given data distribution algorithm is thus a four steps process:

1. **Define the experimental setup** by selecting the software to be used (in special the DBMS), the number of nodes in the cluster, and the TPC-DS scale factor.

2. **Generate the data** using the "dbgen2" utility (Data Generator) of TPC-DS to generate the data and the "qgen2" utility (Query generator) to transform the query templates into executable SQL for the target DBMS. As mentioned before, generated data is temporarily stored in the DWS Data Staging area in a schema similar to the one used in the cluster nodes.

3. **Load the data** into the cluster nodes and measure the data distribution time and the coefficient of variation of the amount of data stored in each node. Due to the obvious non-determinism of the data loading process, this step should be executed (i.e., repeated) at least three times. Ideally, to achieve some statistical representativeness it should be

[1] Notice, however, that low coefficient of variation of the total number of fact rows in each node may not necessarily imply the best response time, as it also depends on the characteristics of the data sent to each node.

executed a much larger number of times; however, as it is a quite heavy step, this may not be practical or even possible. The data distribution time and the CV are calculated as the average of the times and CVs obtained in each execution.

4. **Execute queries** to evaluate the effectiveness of the data placing in terms of the performance of the user queries. TPC-DS queries should be run one at a time and the state of the system should be restarted between consecutive executions (e.g., by performing a cache flush between executions) to obtain execution times for each query that are independent from the queries run before. Due to the non-determinism of the execution time, each query should be executed at least three times. The response time for a given query is the average of the response times obtained for each of the three individual executions.

5. EXPERIMENTAL RESULTS AND ANALYSIS

In this section we present an experimental evaluation of the algorithms discussed in Section 3 using the approach proposed in Section 4. The experiments aim to identify differences, if any, between the various data distribution algorithms, concerning not only the performance of the distribution process itself, but also the performance of the DWS system when accessing data distributed using each of the algorithms.

5.1. Setup and experiments

The basic platform used consist of six Intel Pentium IV servers with 2Gb of memory, a 120Gb SATA hard disk, and running PostgreSQL 8.2 database engine over the Debian Linux Etch operating system. The following configuration parameters were used for PostgreSQL 8.2 database engine in each of the nodes: 950 Mb for shared_buffers, 50 Mb for work_mem and 700 Mb for effective_cache_size.

The servers were connected through a dedicated fast-Ethernet network. Five of them were used as nodes of the DWS cluster, being the other the coordinating node, which runs the middleware that allows client applications to connect to the system, receives queries from the clients, creates and submits the sub queries to the nodes of the cluster, receives the partial results from the nodes and constructs the final result that is sent to the client application.

Two TPC-DS [11] scaling factors were used, 1 and 10, representing initial data warehouse sizes of 1Gb and 10Gb, respectively. These small factors were used due to the limited characteristics of the cluster used (i.e., very low cost nodes) and the short amount of time available to perform the experiments. However, it is important to emphasize, that even with these small datasets it is possible to assess the performance of data distribution algorithms (as we show further on), and preliminary tests (with 100Gb and 1Tb datasets) showed that **the conclusions presented also hold for larger datasets**.

5.2. Data distribution time

The evaluation of the data distribution algorithms started by generating the facts data in the DWS Data Staging Area (DSA), located in the coordinating node. Afterwards, each algorithm was used to compute the destination node for each facts row. Finally, facts rows were distributed to the corresponding nodes. **Table 1** presents the time needed to perform the data distribution using each of the algorithms considered.

The algorithm using a hash function to determine the destination node for each row of the fact tables is clearly the less advantageous. For the 1Gb DW, all other algorithms tested took approximately the same time to populate the star schemas in all nodes of the cluster, with a slight

advantage to round-robin 100 (although the small difference in the results does not allow us to draw any general conclusions). For the 10 Gb DW, the fastest way to distribute the data was using round-robin 1, with an increasing distribution time as a larger window for round-robin is considered. Nevertheless, round-robin 10000, the slowest approach, took only more 936 seconds than round-robin 1 (the fastest), which represents less than 5% extra time.

Table 1. Time (in the format hours:minutes:seconds) to copy the replicated dimension tables and to distribute facts data across the five node DWS system

	Distribution time	
Algorithm	**1 Gb**	**10 Gb**
Random	0:33:16	6:13:31
Round-robin 1	0:32:09	6:07:15
Round-robin 10	0:32:31	6:12:52
Round-robin 100	0:31:44	6:13:21
Round-robin 1000	0:32:14	6:16:35
Round-robin 10000	0:32:26	6:22:51
Hash-based	0:40:00	10:05:43

5.3. Coefficient of variation of the number of rows

Table 2 and **Table 3** display the coefficient of variation of the number of rows sent to each of the five nodes, for each fact table of the TPC-DS schema.

Table 2. CV(\%) of number of rows in the fact tables in each node for the 1Gb data warehouse.

Facts table	**Random**	**RR1**	**RR10**	**RR100**	**RR1000**	**RR10000**	**Hash-based**
catalog_returns	0.70	0.00	0.02	0.18	1.21	8.96	0.64
catalog_sales	0.15	0.00	0.00	0.00	0.00	1.55	0.24
inventory	0.06	0.00	0.00	0.00	0.00	0.10	0.00
store_returns	0.18	0.00	0.01	0.08	0.87	7.53	0.22
store_sales	0.11	0.00	0.00	0.01	0.01	0.94	0.14
web_returns	0.84	0.00	0.03	0.34	3.61	35.73	0.99
web_sales	0.35	0.00	0.00	0.02	0.02	3.79	0.15

Table 3. CV(\%) of number of rows in the fact tables in each node for the 10Gb data warehouse.

Facts table	**Random**	**RR1**	**RR10**	**RR100**	**RR1000**	**RR10000**	**Hash-based**
catalog_returns	0.21	0.00	0.00	0.01	0.15	1.51	0.07
catalog_sales	0.04	0.00	0.00	0.00	0.02	0.10	0.07
inventory	0.02	0.00	0.00	0.00	0.00	0.02	0.00
store_returns	0.08	0.00	0.00	0.00	0.04	0.94	0.12
store_sales	0.05	0.00	0.00	0.00	0.01	0.06	0.08
web_returns	0.18	0.00	0.00	0.03	0.30	3.64	0.20
web_sales	0.12	0.00	0.00	0.00	0.00	0.00	0.01

For both the data warehouses with 1Gb and 10 Gb, the best equilibrium amongst the different nodes in terms of number of rows in each fact table was achieved using round-robin 1. The results obtained for the random and hash-based distributions were similar, particularly for the 1Gb data warehouse.

The values for the CV are slightly lower for 10Gb than when a 1Gb DSA was used, which would be expected considering that the maximum difference in number of rows was maintained but the total number of rows increased considerably.

As the total number of rows in each fact table increases, the coefficient of variation of the number of rows that is sent to each node decreases. If the number of rows to be distributed is considerably small, a larger window for the round-robin distribution will result in a poorer balance. Random and hash-based distributions also yield a better equilibrium of total facts rows in each node if the number of facts rows to distribute is larger.

5.4. Queries response time

To assess the performance of the DWS system during query execution, 27 queries from the TPC Benchmark DS (TPC-DS) were run. The queries were selected based on their intrinsic characteristics and taking into account the changes needed for the queries to be supported by the PostgreSQL DBMS. Note that, as the goal is to evaluate the data distribution algorithms and not to compare the performance of the system with other systems, the subset of queries used is sufficient. The complete set of TPC-DS queries used in the experiments can be found in [33].

5.4.1. Data warehouse of 1Gb

Figure 2 shows the results obtained for five of the TPC-DS queries. As we can see, for some queries the execution time is highly dependent on the data distribution algorithm, while for some other queries the execution time seems to be relatively independent from the data distribution algorithm used to populate each node. The execution times for all the queries used in the experiments can be found at [33].

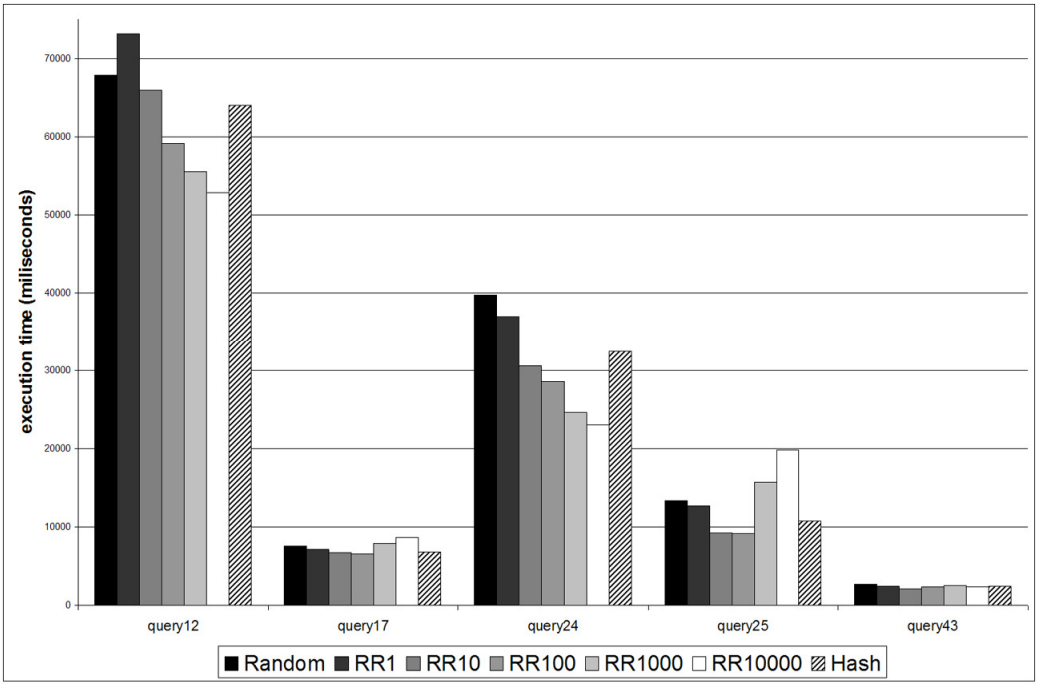

Figure 2. Execution times for each data distribution of a 1Gb data warehouse.

We will focus our discussion of the results on the behavior of queries 24 and 25, as their behavior is representative of the behavior of most of the queries used, highlighting the differences in performance that resulted from using different distribution algorithms. As a first step to understand the results for each query, we analyzed the execution times of the queries in the individual nodes of the cluster. The results for queries 24 and 25 are listed in **Table 4**, along with the mean execution time and the coefficient of variation of the execution times of all nodes.

Table 4. Execution times in each node of the cluster (DW of 1Gb).

		Execution times (ms)						
Query	**Node**	**Random**	**RR1**	**RR10**	**RR100**	**RR1000**	**RR10000**	**Hash**
24	1	31391	30893	28702	24617	19761	20893	27881
	2	28089	30743	29730	24812	20284	3314	27465
	3	38539	35741	29296	23301	20202	3077	29500
	4	31288	29704	29683	24794	23530	6533	31595
	5	35783	33625	28733	27765	21144	21976	30782
CV (%)		12.49	7.72	1.70	6.54	7.19	85.02	6.07
25	1	8336	8519	7775	7426	8293	1798	7603
	2	7073	9094	8338	7794	13763	1457	7109
	3	12349	11620	7523	7885	14584	6011	9022
	4	8882	8428	7175	8117	2927	1533	9732
	5	8782	8666	7561	7457	1881	19034	8621
CV (%)		21.60	14.47	5.59	3.79	71.22	126.57	12.62

By comparing the partial execution times for query 25 (see **Table 4**) to its overall execution time (displayed in **Figure 2**), it is apparent that the greater the unbalance of each node's execution time, the longer the overall execution time of the query. The opposite, though, is observed for query 24: the distribution with the largest unbalance of the cluster nodes' execution times is also the fastest. In fact, although in this case round-robin 10000 presents two clearly slower nodes, they are still faster than the slowest node for any of the other distributions, resulting in a faster overall execution time for the query.

The analysis of the execution plan for query 24 showed that the steps that decisively contribute for the total execution time are three distinct index scans (of the indexes on the primary keys of dimension tables *customer*, *customer_address*, and *item*), executed after retrieving the fact rows from table *web_sales* that comply with a given date constraint (year 2000 and quarter of year 2). Also for query 25, the first step of the execution is retrieving the fact rows from table *catalog_returns* that correspond to year 2001 and month 12, after which four index scans are executed (of the indexes on the primary keys of dimension tables *customer*, *customer_address*, *household_demographics*, and *customer_demographics*).

In both cases, the number of eligible rows (i.e., rows from the fact table that comply with the date constraint) determines the number of times each index is scanned. **Table 5** depicts the number of rows in table *web_sales* and in table *catalog_returns* in each node, for each distribution, that correspond to the date constraints being applied for queries 24 and 25.

As we can observe, the coefficient of variation of the number of eligible facts rows in each node increases as we move from round-robin 1 to round-robin 10000, being similar for random and hash-based distributions. This is a consequence of distributing increasingly larger groups of sequential facts rows from a chronologically ordered set of data to the same node: with the increase of the round-robin "window", more facts rows with the same value for the date key will end up in the same node, resulting in an increasingly uneven distribution (in what concerns the values for that key). In this case, whenever the query being run applies a restriction on the date,

the number of eligible rows in each node will be dramatically different among the nodes for a round-robin 10000 data distribution (which results in some nodes having to do much more processing to obtain a result than others), but more balanced for random or round-robin 1 or 10 distributions.

Table 5. Number of facts rows that comply with the date constraints of queries 24 (table *web_sales*) and 25 (table *catalog_returns*).

		# of facts rows						
Facts Table	**Node**	**Random**	**RR1**	**RR10**	**RR100**	**RR1000**	**RR10000**	**Hash**
web_sales	1	4061	4055	4054	4105	3994	9529	4076
	2	3990	4055	4053	4052	4283	19	3999
	3	4101	4056	4055	4002	3999	7	4044
	4	4042	4056	4056	4002	3998	740	4139
	5	4083	4055	4062	4116	4003	9982	4019
CV (%)		1.06	0.01	0.09	1.34	3.14	128.58	1.35
catalog_ returns	1	489	477	487	499	404	16	483
	2	475	477	470	495	982	6	462
	3	490	477	472	511	960	227	470
	4	468	477	479	457	28	10	484
	5	464	478	478	424	12	2127	487
CV (%)		2.49	0.09	1.40	7.54	100.03	194.26	2.24

Nevertheless, this alone does not account for the results obtained. If that was the case, round-robin 10000 would be the distribution with the poorer performance for both queries 24 and 25, as there would be a significant unbalance of the workload among the nodes, resulting in a longer overall execution time.

The data in **Table 6** sheds some light on why this data distribution yielded a good performance for query 24, but not for query 25. It displays the average time to perform two different index scans: the index scan on the index of the primary key of the dimension table *customer*, executed while running query 24, and the index scan on the index of the primary key of the dimension table *customer_demographics*, executed while running query 25. The number of times each index scan was performed during the execution of the queries and the total number of distinct foreign keys (corresponding to distinct rows in the dimension table) present in the queried fact table, in each node of the system, for round-robin 1 and round-robin 10000 distributions are also displayed.

In both cases, the average time to perform the index scan on the index over the primary key of the dimension table in each of the nodes was very similar for round-robin 1, but quite variable for round-robin 10000. In fact, during the execution of query 24, the index scan on the index over the primary key of the table *customer* was quite fast in nodes 1 and 5 for the round-robin 10000 distribution and, in spite of having the largest number of eligible rows in those nodes, they ended up executing faster than all the nodes for the round-robin 1 distribution. Although there seems to be some preparation time for the execution of an index scan, independently of the number of rows that are afterwards looked for in the index (which accounts for the higher average time for nodes 2, 3 and 4), carefully looking at the data on **Table 6** allows us to conclude that the time needed to perform the index scan in the different nodes decreases when the number of distinct primary key values of the dimension that are present in the fact table scanned also decreases.

Table 6. Average time to perform an index scan on dimension table *customer* (query 24) and on dimension table *customer_demographics* (query 25).

Query	Algorithm	Node	Exec. Time (ms)	Index scan on dimension		Diff. values of foreign key in facts table
				avg time (ms)	# of times perf.	
24	Round-robin 1	1	30893	3.027	4055	42475
		2	30743	3.074	4055	42518
		3	35741	3.696	4056	42414
		4	29704	2.858	4056	42458
		5	33625	3.363	4055	42419
	Round-robin 10000	1	20893	0.777	9529	12766
		2	3314	17.000	19	12953
		3	3077	19.370	7	12422
		4	6533	1.596	740	12280
		5	21976	0.725	9982	12447
25	Round-robin 1	1	8519	0.019	38	28006
		2	9094	0.019	32	27983
		3	11620	0.019	22	28035
		4	8428	0.019	32	28034
		5	8666	0.017	39	28035
	Round-robin 10000	1	1798	66.523	1	29231
		2	1457	73.488	1	29077
		3	6011	34.751	2	29163
		4	1533	-	0	29068
		5	19034	19.696	146	23454

This way, the relation between the number of distinct values for the foreign keys and the execution time in each node seams to be quite clear: the less distinct keys there are to look for in the indexes, the shorter is the execution time of the query in the node (mostly because the less distinct rows of the dimension that need to be looked for, the less pages need to be fetched from disk, which dramatically lowers I/O time). This explains why query 24 runs faster in a round-robin 10000 data distribution: each node had fewer distinct values of the foreign key in the queried fact table. For query 25, as the total different values of foreign key in the queried fact table in each node was very similar, the predominant effect was the unbalance of eligible rows, and round-robin 10000 data distribution resulted in a poorer performance.

These results ended up revealing an crucial aspect: some amount of clustering of fact tables rows, concerning each of the foreign keys, seems to result in an improvement of performance (as happened for query 24), but too much clustering, when specific filters are applied to the values of that keys, result in a decrease of performance (as happened for query 25).

5.4.2. Data warehouse of 10Gb

The same kind of results were obtained for a DWS system equivalent to a 10Gb data warehouse, and the 3 previously identified behaviors were also found: queries whose execution times did not appear to depend on the distribution, queries that ran faster on round-robin 10000, and queries that ran faster on the random distribution. The queries presented more balanced execution times for all data distributions, the exception being round-robin 10000, for which most queries had either the shortest or the longest execution time.

Query 25, for example, had now more even execution times for all data distributions, except for round-robin 10000, in which it was still noticeably slower. With the considerable increase of the

total number of rows in each facts table, the difference in number of eligible facts rows (after applying the date constraint) between the nodes became quite small for random and round-robin with smaller "windows", and the effect of uneven work among the nodes was toned down for these distributions, therefore having a significant impact on total execution time only for round-robin 10000. This is made clear by the data on **Table 7**, and further confirms the findings discussed in the previous section.

Table 7. Execution times and number of facts rows in table *catalog_returns* that comply with the date constraint of query 25, for data warehouses of 1Gb and 10Gb.

Algorithm	Node	10Gb		1Gb	
		Ex. Time (ms)	# of elegible facts rows	Ex. Time (ms)	# of elegible facts rows
Random	1	51332	4801	8336	489
	2	59553	4720	7073	475
	3	65959	4858	12349	490
	4	63130	4741	8882	468
	5	57764	4691	8782	464
CV (%)		9.37	1.41	21.60	2.49
Round-robin 1000	1	65303	5000	8293	404
	2	63286	5000	13763	982
	3	47439	4255	14584	960
	4	59298	4556	2927	28
	5	58479	5000	1881	12
CV (%)		11.79	7.19	71.22	100.03
Round-robin 10000	1	23845	1556	1798	16
	2	90823	10000	1457	6
	3	103885	10000	6011	227
	4	38238	2255	1533	10
	5	2303	0	19034	2127
CV (%)		84.40	101.86	126.57	194.26

The larger the number of facts rows to distribute among the nodes, the more round-robin data distributions with small "windows" (relative to the total number of facts rows being distributed) resemble the random distribution.

Other queries had also leveled execution times for each of the data distributions, but were still clearly faster for round-robin 1000 and 10000 distributions, as was the case for queries 12 and 24. In fact, the time advantage for these distributions has been dilated, as now the unbalance of eligible facts rows in each node has been diminished, and the predominant effect becomes the lower number of distinct dimension keys in each node for the round-robin distributions with the largest windows (see data on **Table 8**).

When the number of facts rows to distribute is small, a random distribution is not as effective as with a larger number of rows in obtaining an even distribution of foreign keys (either date dimension keys, which significantly influence the number of eligible rows, or the total number of distinct keys of other dimensions) in each node. In the case of a 10Gb data warehouse, as the amount of data was significantly higher, the random distribution caused better spreading of the data than the round-robin with small windows caused in the 1Gb distribution.

Table 8. Execution times, number of facts rows in table *web_sales* that comply with the date constraint of query 24, and corresponding number of different foreign keys for dimension *customer*.

Algorithm	Node	Ex. Time (ms)	# of elegible facts rows	Diff. values of foreign key
Random	1	219359	41010	334469
	2	251610	40932	334316
	3	232501	41442	334618
	4	231936	41139	334394
	5	247064	41329	334678
CV (%)		5.47	0.52	0.05
Round-robin 1000	1	219961	41200	334139
	2	220745	41133	334123
	3	258224	41119	334166
	4	253699	41200	334032
	5	254747	41200	334299
CV (%)		8.02	0.10	0.03
Round-robin 10000	1	132622	40000	143899
	2	129446	40000	145687
	3	146040	40000	145875
	4	165073	45852	145477
	5	150075	40000	146115
CV (%)		9.92	6.36	0.60

Round-robin 1 is always a bad distribution in what concerns the spreading of the different dimension keys, as it tends to send to every node nearly every distinct key possible. With the increase of the total number of rows in each facts table, random distribution became more effective, and with even more initial data to distribute, it would probably end up performing better than round-robin 10000. Still, there will always be a window for which the distribution of different dimension keys to each node is more effective than the random distribution.

But even though the best distribution was not the same for the 10Gb data warehouse, the reason for it is similar: eligible rows for queries were better distributed among the nodes and lower number of distinct primary keys values of the dimension on the fact tables determined the differences.

6. CONCLUSIONS AND FUTURE WORK

This work analyzes three data distribution algorithms for the loading of the nodes of a data warehouse using the DWS technique: random, round-robin and a hash-based algorithm. Overall, the most important aspects we were able to draw from the experiments were concerning two values: 1) the number of distinct values of a particular dimension within a queried fact table and 2) the number of rows that are retrieved after applying a particular filter in each node.

As a way to understand these aspects, consider, for instance, the existence of a data warehouse with a single fact table and a single dimension, constituted by 10000 facts corresponding to 100 different dimension values (100 rows for each dimension value). Consider, also, that we have the data ordered by the dimension column and that there are 5 nodes. There are two opposing distributions possible, which distribute evenly the rows among the five nodes (resulting 2000 rows in each node): a typical round-robin 1 distribution that copies one row to each node at a time, and a simpler one that copies the first 2000 rows to the first node, the next 2000 to the second, and so on.

In the first case, all 100 different dimension values end up in the fact table of each node, while, in the second case, the 2000 rows in each node have only 20 of the distinct dimension values. As consequence, a query execution on the first distribution may imply the loading of 100% of the dimension table in all of the nodes, while on the second distribution a maximum of 20% of the dimension table will have to be loaded in each node, because each node has only 20% of all the possible distinct values of the dimension.

If the query run retrieves a large number of rows, regardless of their location on the nodes, the second distribution would result in a better performance, as fewer dimension rows would need to be read and processed in each node. On the other hand, if the query has a very restrictive filter, selecting only a few different values of the dimension, then the first distribution will yield a better execution time, because these different values will be more evenly distributed among the nodes, resulting in a more distributed processing time, thus lowering the overall execution time for the query.

The aforementioned effects suggest an optimal solution to the problem of the loading of the DWS. As a first step, this loading algorithm would classify all the dimensions in the data warehouse as large dimensions and small dimensions. Exactly how this classification would be done depends on the business considered (i.e., on the queries performed) and must also account the fact that this classification might be affected by subsequent data loadings. The effective loading of the nodes must then consider complementary effects: it should minimize the number of distinct keys of any large dimension in the fact tables of each node, minimizing the disk reading on the nodes and, at the same time, it should try to split correlated rows among the nodes, avoiding that eligible rows of typical filters used in the queries end up grouped in a few of them.

However, to accomplish that, it appears to be impossible to decide beforehand a specific loading strategy to use without taking the business into consideration. The suggestion here would be to analyze the types of queries and filters mostly used in order to decide what would be the best solution for each case.

REFERENCES

[1] R. Kimball and M. Ross, The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd ed., no. 0. John Wiley & Sons, Inc., 2002, p. 464.

[2] R. Kimball and J. Caserta, The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons, Inc., 2004, p. 528.

[3] L. Agosta, "Data warehousing lessons learned: SMP or MPP for data warehousing," Information Management Magazine, May-2002.

[4] Sun Microsystems, "Data Warehousing Performance with SMP and MPP Architectures, Technical White Paper," 1997.

[5] "Netezza, an IBM Company." [Online]. Available: http://www.netezza.com/.

[6] "Greenplum." [Online]. Available: http://www.greenplum.com/.

[7] J. Bernardino and H. Madeira, "Experimental Evaluation of a New Distributed Partitioning Technique for Data Warehouses," in Proceedings of the International Symposium on Database Engineering and Applications (IDEAS01), 2001, pp. 312-321.

[8] J. R. Bernardino, P. S. Furtado, and H. C. Madeira, "Approximate Query Answering Using Data Warehouse Striping," Journal of Intelligent Information Systems, Special issue on Data Warehousing and Knowledge Discovery, vol. 19, no. 2, pp. 145-167, Sep. 2002.

[9] "Critical Software S.A." [Online]. Available: http://www.criticalsoftware.com.

[10] R. Almeida, J. Vieira, M. Vieira, H. Madeira, and J. Bernardino, "Efficient Data Distribution for DWS," in Proceedings of the 10th international conference on Data Warehousing and Knowledge Discover (DaWaK'08), 2008, vol. 5182, pp. 75-86.

[11] Transaction Processing Performance Council (TPC), "TPC Benchmark DS (decision support) standard specification, draft version 32," 2005. [Online]. Available: www.tpc.org/tpcds.

[12] Hongjun Lu, Query Processing in Parallel Relational Database Systems. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994.
[13] M. T. Özsu and P. Valduriez, Principles of distributed database systems, 2nd ed. Prentice-Hall, Inc., 1999.
[14] P. Mohankumar, P. Kumaresan, and J. Vaideeswaran, "Optimism analysis of parallel queries in databases through multicores," International Journal of Database Management Systems (IJDMS), vol. 3, no. 1, pp. 156-164, 2011.
[15] H. Garcia-Molina, W. J. Labio, J. L. Wiener, and Y. Zhuge, "Distributed and Parallel Computing Issues in Data Warehousing," in Proceedings of the 17th annual ACM Symposium on Principles of Distributed Computing (PODC'98), 1998, p. 7.
[16] Y. J. Singh, Y. S. Singh, A. Gaikwad, and S. C. Mehrotra, "Dynamic management of transactions in distributed real-time processing system," International Journal of Database Management Systems (IJDMS), vol. 2, no. 2, pp. 161-170, 2010.
[17] P. P. Karde and V. M. Thakare, "Selection Of Materialized View Using Query Optimization In Database Management : An Efficient Methodology," International Journal of Database Management Systems (IJDMS), vol. 2, no. 4, pp. 116-130, 2010.
[18] Oracle, "Oracle® Database Data Warehousing Guide 11g Release 2 (11.2)," 2011. [Online]. Available: http://docs.oracle.com/cd/E11882_01/server.112/e25554.pdf.
[19] IBM, "IBM Red Brick Warehouse." [Online]. Available: http://www-01.ibm.com/software/data/informix/redbrick/.
[20] C. A. Galindo-Legaria et al., "Optimizing Star Join Queries for Data Warehousing in Microsoft SQL Server," in Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE'08), 2008, pp. 1190-1199.
[21] T. Stöhr, H. Märtens, and E. Rahm, "Multi-Dimensional Database Allocation for Parallel Data Warehouses," in Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00), 2000, pp. 273-284.
[22] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," ACM SIGMOD Record, vol. 26, no. 1, pp. 65-74, Mar. 1997.
[23] J. Albrecht, H. Guenzel, and W. Lehner, "An Architecture for Distributed OLAP," in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), 1998.
[24] K.-D. Schewe and J. Zhao, "Balancing Redundancy and Query Costs in Distributed Data Warehouses: an approach based on abstract state machines," in Proceedings of the 2nd Asia-Pacific conference on Conceptual Modelling (APCCM '05), 2005, pp. 97-105.
[25] B. Liu, S. Chen, and E. A. Rundensteiner, "A Transactional Approach to Parallel Data Warehouse Maintenance," in Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'02), pp. 307–317.
[26] M. O. Akinde, M. H. Böhlen, T. Johnson, L. V. S. Lakshmanan, and D. Srivastava, "Efficient OLAP Query Processing in Distributed Data Warehouses," Information Systems - Special issue: Best papers from EDBT 2002, vol. 28, no. 1-2, pp. 111-135, Mar. 2003.
[27] I. Stanoi, D. Agrawal, and A. E. Abbadi, "Modeling and Maintaining Multi-View Data Warehouses," in Proceedings of the 18th International Conference on Conceptual Modeling (ER'99), 1999, pp. 161-175.
[28] D. Kossmann, M. J. Franklin, G. Drasch, and W. Ag, "Cache investment: integrating query optimization and distributed data placement," ACM Transactions on Database Systems, vol. 25, no. 4, pp. 517-558, Dec. 2000.
[29] R. Vingralek, Y. Breitbart, and G. Weikum, "Distributed file organization with scalable cost/performance," ACM SIGMOD Record, vol. 23, no. 2, pp. 253-264, Jun. 1994.
[30] M. Costa and H. Madeira, "Handling big dimensions in distributed data warehouses using the DWS technique," in Proceedings of the 7th ACM international workshop on Data Warehousing and OLAP (DOLAP'04), 2004, p. 31.
[31] J. Vieira, J. Bernardino, and H. Madeira, "Efficient compression of text attributes of data warehouse dimensions," in Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery (DaWak'05), 2005.
[32] B. Jenkins, "Hash Functions," Algorithm Alley, Dr. Dobb's, 1997. [Online]. Available: http://drdobbs.com/database/184410284. [Accessed: 22-Nov-2011].
[33] R. Almeida and M. Vieira, "Selected TPC-DS queries and execution times," 2008. [Online]. Available: http://eden.dei.uc.pt/~mvieira/. [Accessed: 22-Nov-2011].

Authors

Raquel Almeida is a Ph. D. student at the University of Coimbra. She received her B.Sc. Degree in Multimedia and Communications at the Department of Informatics Engineering of the University of Coimbra in 2007. In 2009 started her Ph. D. in Information Science and Technology also in the University of Coimbra. Since 2007 she has been with the Centre for Informatics and Systems of the University of Coimbra (CISUC), researching topics related to affordable data warehouses and benchmarking resilience of self-adaptive systems. She has authored or co-authored 6 papers in refereed conferences.

Jorge Vieira is the manager for the Database and Business Intelligence unit at Critical Software, an international software company headquartered in Coimbra, Portugal, with offices in Lisbon, Porto, San Jose (US), London (UK), Romania (RO), Brazil (BR), Mozambique (MZ) and Angola (AO). In 2002 got a degree in Computer Science by the University of Coimbra and enrolled on a Post-Graduation in Decision Support Systems. From 2003 to 2005 he worked on R&D projects in cooperation with Coimbra University. During this period he published several papers in international conferences. From 2005 to 2009 he has been the technical manager of several projects in Energy, Government, Telecommunications and Manufacturing markets. He was responsible for technical leading of some of the most important projects developed on those areas by Critical Software.

Marco Vieira is an Assistant Professor at the University of Coimbra, Portugal. He is an expert on dependability and security benchmarking and his research interests also include experimental dependability evaluation, fault injection, data warehousing, software development processes, and software quality assurance, subjects in which he has authored or co-authored more than 100 papers in refereed conferences and journals. He has participated in many research projects, both at the national and European level. Marco Vieira has served on program committees of the major conferences of the dependability and databases areas and acted as referee for many international conferences and journals in those areas.

Henrique Madeira is a full professor at the University of Coimbra, where he has been involved in the research on dependable computing since 1987. He has authored or co-authored more than 140 papers in refereed conferences and journals and has coordinated or participated in tens of projects funded by the Portuguese government and by the European Union. He was the Program Co-Chair of the International Performance and Dependability Symposium track of the IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-PDS2004 and was appointed Conference Coordinator of IEEE/IFIP DSN 2008.

Jorge Bernardino is Coordinator Professor of the Department of Systems and Computer Engineering at the ISEC (Engineering Institute of Coimbra) of Polytechnic of Coimbra, Portugal. He was President of ISEC from 2005–2010 and President of Scientific Council during 2003–2005. He received his PhD Degree in Computer Engineering from the Computer Engineering Department of the University of Coimbra in 2002. He is member of Software and Systems Engineering (SSE) group of Centre for Informatics and Systems of the University of Coimbra (CISUC) research centre. His main research areas are Data Warehousing, Business Intelligence, Database Knowledge Management, e-business and e-learning.