An Open Source System for Big Data Warehousing

Nunziato Cassavia[1,2], Elio Masciari[1] and Domenico Saccà[2,3]
[1]*ICAR-CNR, Rende, Italy*
[2]*DIMES UNICAL, Rende Italy*
[3]*Centro di Competenza ICT-SUD, Rende, Italy*

Keywords: Big Data Warehousing, NoSQL and Mondrian.

Abstract: The pervasive diffusion of new data generation devices has recently caused the generation of massive data flows containing heterogeneous information generated at different rates and having different formats. These data are referred as *Big Data* and require new storage and analysis approaches to be investigated for managing them. In this paper we will describe a system for dealing with massive big data stores. We defined an open source tool that exploits a NoSQL approach for data warehousing in order to offer user am intuitive way to easily query data that could be quite hard to be understood otherwise.

1 INTRODUCTION

The increasing availability of huge amounts of data from heterogeneous sources, calls for the definition of new paradigms for their management – this problem is known with the name *Big Data* (Nature, 2008; Economist, 2010; Economist, 2011; Agrawal et al., 2012; Lohr, 2012; Manyika et al., 2011; Noguchi, 2011a; Noguchi, 2011b; Labrinidis and Jagadish, 2012). As a consequence of new perspective on data, many traditional approaches to data analysis result inadequate both for their limited effectiveness and for the inefficiency in the management of the huge amount of available information. Therefore, it is necessary to rethink both the storage and access patterns to big data as well the design of new tools for data presentation and analysis. In particular, On Line Analytical Processing (OLAP) tools require suitable adjustments in order to work for big data processing effectively. Indeed, it is crucial, during the construction and analysis of a data warehouse, to exploit ad-hoc tools that allow an easy and fast search of data stored in several nodes distributed over the storage layer.

More in detail, while building a data warehouse for Big Data, the key to a successful analysis (i.e. a fast and effective one) is the availability of good indexing mechanisms. Therefore, an additional cost in terms of storage space consumption needed for storing the appropriate indices is to be taken into account.

It is worth noticing that the problem of fast accessing relevant pieces of information arises in several scenarios such as world wide web search, e-commerce systems, mobile systems and social networks analysis to cite a few.

Successful analyses for all the application contexts rely on the availability of effective and efficient tools for browsing data so that users may eventually extract new knowledge which s/he was not interested initially.

In this paper, we describe the architecture of an open source system for big data OLAP, capable to deal with Big Data and offering the chance to "navigate through" the data in a simplified manner, while keeping traditional operators available in an OLAP based system such as roll-up, drill-down, slice and dice. Our system is based on the reliable and widely used MonetDB NOSQL data store and the Mondrian analysis tool (thus we named it MonOLAP) as will be described in next sections.

2 MonOLAP HIGH LEVEL SYSTEM DESIGN

In this section, we will describe our system design. As our goal is the development of an Open Source tool that takes advantage of both Mondrian and MonetDB systems, we leverage the abstraction features offered by Mondrian and the data storage layer functionalities of MonetDB. Our system assumes that the target data could be stored on some external relational

Cassavia, N., Masciari, E. and Saccà, D.
An Open Source System for Big Data Warehousing.
DOI: 10.5220/0006485703060313
In *Proceedings of the 6th International Conference on Data Science, Technology and Applications (DATA 2017)*, pages 306-313
ISBN: 978-989-758-255-4
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

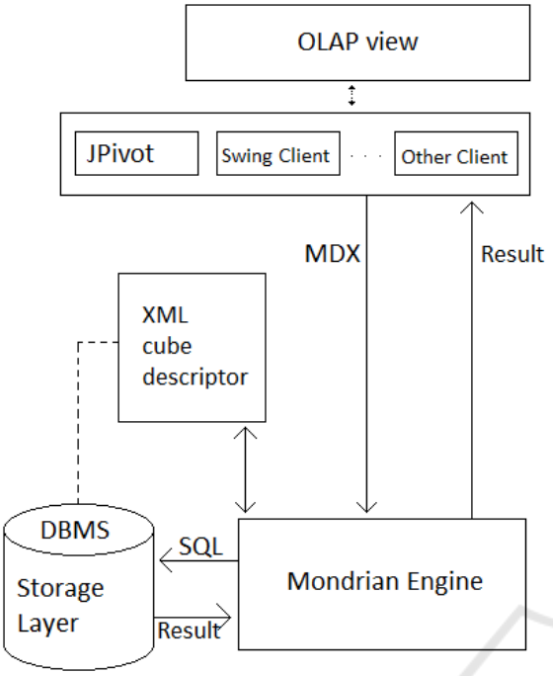

Figure 1: Mondrian Architecture.

DBMS. Thus, the system allows the user to select her own interesting data from relational datasources and the possibility to create, from the selected data, her custom multidimensional datacubes.

In order to load the data into the MonetDB repository, it is mandatory to perform a transformation and migration step. Moreover, the definition of multidimensional datacube will be tailored to support the same multidimensional view on MonetDB own data structures. The transformation of the original data affects the materialization of the datacube as well as data flattening. The latter step involves the denormalization of the OLAP schema fact table, i.e., we add to the fact table the attributes related to the measure of interest declared in the OLAP schema.

The outcome of this preliminary operation is a performance improvement, as the increase in the number of attributes in the fact table will perfectly fit the MonetDB column management strategy. The performance improvement obtained by this synergy between the two systems will be clearer from the experimental evaluation shown in next sections. Figure 1 shows the block diagram of the architecture of Mondrian OLAP system.

We briefly summarize here the main Mondrian features we exploited in our prototype implementation:

- the system allows data cube querying using MDX language by a simple user interface. In order to properly perform the queries, the definition of an XML file representing the datacube and its dimensions, hierarchies and measures, is mandatory;
- the MDX query is forwarded to the OLAP engine that perform the transformation to SQL and execute the queries to the repository containing the data;
- the query result is sent to Mondrian for user defined processing and then sent to the user interface which displays the results in a multidimensional way using interfacing-dependent mechanisms.

In Figure 2 we report the overall architecture of our MonOLAP engine. It contains several modules that have been used to exploit MonetDB features to create a high-performance still reliable system. More in detail, two modules plays a crucial role in the system architecture, namely the ETL tool and the DBMS MonetDB which are responsible respectively for manipulating the data and storing the data in flattened form.

The ETL tool deals with:

- extracting data from the data source (that we recall could be an external DBMS);
- materializing the OLAP cube;
- transforming the cube by flattening it;
- transferring the data to the MonetDB storage layer.

It is worth noticing that, the configuration file that defines the cube dimensions, measures, and layers (that refers to the original DBMS) has to be built according to the warehouse architect specification for the domain being analyzed. The obtained XML file, defined as *Adapted Cube Descriptor*, is the result of a transformation that takes into account the new flattened schema for data materialization.

For the sake of completeness, we report in Figure 3 the workflow for OLAP cube definition that is performed by user using the Schema Workbench tool.

3 MonOLAP MODULES DESCRIPTION

As explained above, our system is an integrated environment for supporting OLAP data analysis for Big Data. The novelty of our approach mainly consists in its easy features for data manipulation and their transfer to/from any Relational DBMS selected by the user. However, after an extensive set of experiments, we found that optimal performances can be obtained by leveraging MonetDB. We also manage the data mapping phase in order to leave the multidimensional

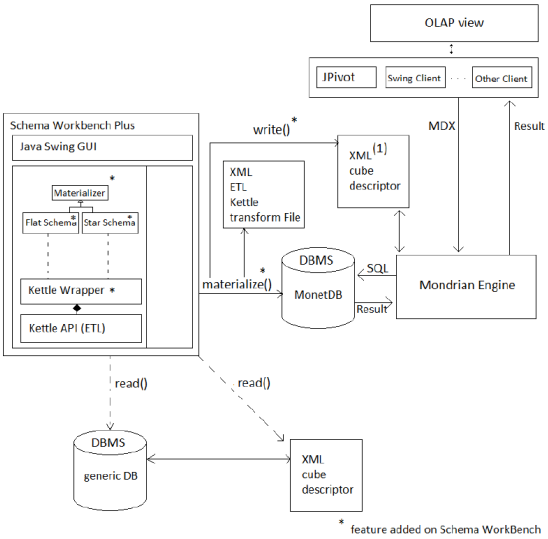

Figure 2: MonOLAP Engine Architecture.

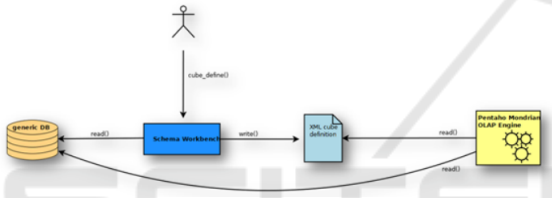

Figure 3: OLAP cube definition by Schema Workbench.

view unchanged while using the MonetDB data structures. This results in a high performance gain due to the fact that MonetDB is extremely efficient in managing tables with a large number of attributes. Based on this preliminary observation, the name *MonOLAP* was chosen to recall its peculiar features deriving from the integration of Mondrian with MonetDB for OLAP analysis ((Mon)drian + (Mon)etDB for OLAP = MonOLAP).

As introduced in the high level description of the system, the MonOLAP System consists of 3 software modules:

- MonetDB;
- Mondrian;
- Schema Workbench Plus.

While MonetDB and Mondrian are respectively the Storage Layer and OLAP Server of the system, the last module, Schema Workbench Plus, can be defined as a *cube designer* tailored for date cube definition and several forms of materialization through ETL mechanisms.

More in detail, the main features of the above mentioned modules, that have been extensively used can be summarized as follows:

- **Schema Workbench Plus.**

  Creation or loading of the XML schema representing a cube: after defining the connection to the source DBMS, the user can create and validate an XML schema for a given cube or alternatively can also update an existing schema. The validation of the schema is performed by interacting with the data repository (by checking the match of the fact table and dimensions with the input data sources);

  Materialization of data cube on the storage layer: the user, after selecting one of the cubes of the schema, can perform the materialization on the storage layer. The system offers the user the possibility to choose, depending on her needs, the type of materialization.

- **Mondrian.**

  After configuring Mondrian configuration parameters, i.e. the connection to the DBMS and the XML schema path, the user can query the data cube in graphical mode and then obtain a multidimensional view of the data. Schema Workbench Plus plays a crucial role at this stage, since it is completely transparent to the user who is able to generate a XML scheme of the materialized cube which allows the query engine to answer to the same MDX queries of the original data cube.

The cube data description is stored by Schema Workbench Plus in XML format. The materialization of the cube causes the execution of two operations:

- the materialization on the storage layer, which can be of two types, flattened or starred;
- the generation of the XML schema for the materialized cube.

4 MATERIALIZATION ALGORITHM

In this section we will explore the implemented materialization techniques.

4.1 Flat Schema

Algorithm 1 illustrates the materialization algorithm for a Flat Schema. The algorithm takes as input two parameters, i.e., the data cube to be materialized (C) and the table for materializing the datacube (F).

The materialization step requires a preliminary check on the type of table of the data cube. When defining an OLAP data cube, user has to specify the fact table for measure computation. The peculiarity

Algorithm 1 Flat Schema Algorithm.

```
1:  procedure FLAT SCHEMA
2:      C ← selected cube
3:      F ← flatten table
4:      if C.factTable is a View then
5:          calculate fact table from SQL expression
6:          calculate fact attribute from SQL expression
7:      else
8:           get fact table from cube
9:      for m in C.measure do
10:         if m is SQL Expression then
11:             Calculate Attribute
12:             Add Attribute to Flatten Table
13:         else
14:             Add Attribute to Flatten Table
15:     for d in C.dimension do
16:         Select d attributes from dimension table
17:         F• join(fact table, dimension table)
```

of this table relies on the fact that it can be either a table on the database or a multi-table view. In the case of a multi-table view, we need to extrapolate the fact table by keeping track of the attributes to be selected because they will then be routed to the flatten table. Once the fact table has been defined, we need to check how the user defined the cube dimensions. Measures on a given cube may refer to a specific column of the fact table or it may be the result of an SQL expression that involves multiple columns in the fact table. Then, before adding the attribute measure to the flatten table, it is necessary to compute the SQL expression . Finally, it is necessary to tweak all the dimensions of the data cube by performing the join operation between the fact table and the dimension table in order to update the flatten table with all the columns that affect the chosen dimensions.

4.1.1 Star Schema

Star schema materialization can be performed by applying flat materialization to each dimension. In Algorithm 2, we report the preudo code of this algorithm.

As in the previous case, the key operation to be performed is fact table and measures identification, and then the partial materialization can take place as he fact table that contains all measures and all external reference keys to the dimension tables. A new table is created for each dimension, which contains all the attributes of interest for the dimension being materialized.

Algorithm 2 Star Schema Algorithm.

```
1:  procedure STAR SCHEMA
2:      C ← selected cube
3:      T ← fact table
4:      if T is a View then
5:          calculate fact table from SQL expression
6:          calculate fact attribute from SQL expression
7:          add all foreign keys to fact table
8:          materialize fact table
9:      else
10:         get fact table from cube
11:         add all foreign keys to fact table
12:         materialize fact table
13:     for m in C.measure do
14:         if m is SQL Expression then
15:             Calculate Attribute
16:             Add Attribute to Fact Table
17:         else
18:             Add Attribute to Fact Table
19:     for d in C.dimension do
20:         Select d attributes from dimension table
21:         Create new table D
22:         D• join(fact table, dimension table)
```

5 EXPERIMENTAL EVALUATION ON TETRA NETWORK

In this section, we describe the results we obtained by running MonOLAP on TETRA network data log. TETRA (*TErrestrial Trunked RAdio*, originally Trans-European Trunked RAdio) is a commercial radio waveform standard with mobile and portable systems used mainly by public security, military agencies and emergency services as well as by private civil services. TETRA is a set of standards for private telecommunication systems targeted to a professional user, but also for service providers interested in having their own radio network.

TETRA is the first open standard for professional digital radio systems and its services are provided with a whole range of NE elements (Network Elements) that supply the management system and support real-time operation and maintenance, necessary to guarantee the daily operation of the system. TETRA provides a communication system for several usage scenarios. A call log is generated for each generated call within the TETRA Network. This call log provides call details, including the parties involved in the call, their location, call duration, and resource usage. For many activities, the full log of a call is built by a set of small call logs that are generated by the various Control Nodes within network. A context log is

generated for each data packet that can provide information about activating and deactivating a data packet and the amount of data packets that are transferred in both directions between the mobile and the Control Node.

5.1 Call Logs Database on TETRA Network

The database contain the real logs generated on TETRA network in the Emilia Romagna region. This database is a relational database of more than 30GB size containing multiple information on each tethered call. The database was built using the Firebird RDBMS and has the following features:

- 124 tables;
- 2000 attributes ;
- 20 tables for storing log information;
- 70 attributes for each log table (on average);
- 30 Gigabytes of data, of disk occupation;
- 132.165.089 tuples.

We have first identified a fact of interest in order to define a Data Mart for OLAP analysis. We have chosen to locate logs for calls that generated alerts, that is, those for which the call was interrupted or disconnected for external cause.

5.2 Logical Model of the OLAP Cube

The source database is very heterogeneous and contains many unnecessary data for the analysis to be carried out. For this reason, it was crucial, through ETL procedures, to extract the data log of interest. In order to perform these transformations, it is necessary to define in the measures of interest, dimensions and hierarchies that we want to analyze. The conceptual model used to represent the information schema is the Dimensional Fact Model, due to some privacy constraints, we limited our analysis to **number of calls** measures.

As regards the analysis dimensions, we identified the following:

- **CALL END REASON** (cardinality 23)**.** Indicates the reason for call ending, this type of information is already classified in the original database in a table that lists 23 reasons why a TETRA network call may end;
- **DISCONNECT CAUSE** (cardinality 203)**.** indicates the reason why an user has been disconnected from the network, differently from the reason for ending a call, this dimension states why it is disconnected from the TETRA network, it contains 203 causes of disconnection;

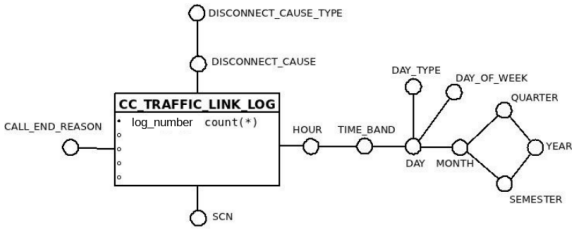

Figure 4: Dimensional Fact Model for OLAP Analisys on Tetra Network.

- **SCN** (cardinality 2)**.** as the data comes from a log database on TETRA Network of Emilia Romagna, there is no reason to have a territorial dimension, except for the SCN to which the interrupted call refers. Analysis of the source database we found that all data refers only to two SCNs: one located in the municipality of Bologna and one in the municipality of Faenza;
- **TEMPORARY DIMENSION** (cardinality 8760 as this number refers to the hours in one year)**.** logs are stored in detail per millisecond, but there is no practical need to aggregate the data till such finer granularity then we decided to support the analysis of the temporal dimension hierarchically by hours, days and months. The year does not make sense in our experimental analysis as the data refer to 2010. Also we want to be able to aggregate for: days in week and weekend, day of the week, time zone and day type. By analyzing the data based on this model we will be able to understand on which days network problems occurred, for what reasons user have been disconnected, for what reasons the calls have been terminated and on what SCN, for which time slot, for which days of the week and so on.

The Dimensional Fact Model referring to the data model for this OLAP analysis is shown in 4.

Using ETL procedures, the table of facts and dimensions was extracted from the original database and uploaded also to MySQL DBMS for the sake of comparison. The Star Schema, representing the DFM for TETRA logs on MySQL, is shown in 5

5.2.1 Experimental Settings

Using MonOLAP, with the Schema Workbench PLUS component on the schema described in Figure 5, the following schemes are obtained:

- Star Schema on MySQL
- Flat Schema on MonetDB

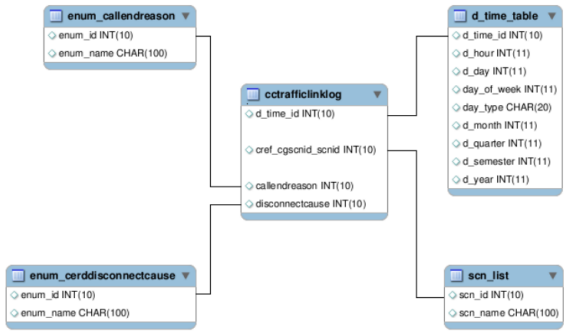

Figure 5: Star Schema for OLAP Analisys on Tetra Network.

- Star Schema on MonetDB
- Flat Schema on MySQL

Our test bench is composed by the following queries:

- Query 1. Drill Down on SCN dimension
- Query 2. Drill Down on dimensions SCN and CALL END REASON
- Query 3. Drill Down on dimensions CALL END REASONS and DISCONNECT CAUSE
- Query 4. Drill Down on TIME dimension
- Query 5. Drill Down on dimensions TIME and DISCONNECT CAUSE
- Query 6. Drill Down on SCN and TIME dimensions

In the following, we report the queries on the OLAP cube expressed in MDX Language:

Query 1.

```
select {[Measures].[numerolog]}
ON COLUMNS,
Crossjoin
(Hierarchize(Union({[scn].[all]},
[scn].[all].Children)),
{([callendreason].[all],
[disconnectcause].[all],
[time.tempo_standard].[all])})
ON ROWS
from [traffic]
```

Query 2.

```
select {[Measures].[numerolog]}
ON COLUMNS,
Crossjoin
(Hierarchize
(Union(Union(Crossjoin({[scn].[all]},
{[callendreason].[all]}),
Crossjoin({[scn].[all]},
[callendreason].[all].Children)),
Union(Crossjoin([scn].[all].Children,
{[callendreason].[all]}),
Crossjoin([scn].[all].Children,
[callendreason].[all].Children)))),
{([disconnectcause].[all],
[time.tempo_standard].[all])})
ON ROWS
from [traffic]
```

Query 3.

```
select {[Measures].[numerolog]}
ON COLUMNS,
Hierarchize(Union(Crossjoin({[scn].[all]},
Union(Crossjoin({[callendreason].[all]},
{([disconnectcause].[all],
[time.tempo_standard].[all])}),
Crossjoin({[callendreason].[all]},
Crossjoin([disconnectcause].[all].Children,
{[time.tempo_standard].[all]})))),
Crossjoin({[scn].[all]},
Crossjoin([callendreason].[all].Children,
{([disconnectcause].[all],
[time.tempo_standard].[all])}))))
ON ROWS
from [traffic]
```

Query 4.

```
select {[Measures].[numerolog]}
ON COLUMNS,
Hierarchize
(Crossjoin({[scn].[all]},
Crossjoin({[callendreason].[all]},
Union(Crossjoin({[disconnectcause].[all]},
{[time.tempo_standard].[all]}),
Crossjoin({[disconnectcause].[all]},
[time.tempo_standard].[all].Children)))))
ON ROWS
from [traffic]
```

Query 5.

```
select {[Measures].[numerolog]}
ON COLUMNS,
Hierarchize
(Crossjoin({[scn].[all]},
Union(Crossjoin({[callendreason].[all]},
Union(Crossjoin({[disconnectcause].[all]},
{[time.tempo_standard].[all]}),
Crossjoin({[disconnectcause].[all]},
[time.tempo_standard].[all].Children))),
Crossjoin({[callendreason].[all]},
Union(Crossjoin([disconnectcause].[all].
Children,{[time.tempo_standard].[all]}),
Crossjoin([disconnectcause].[all].Children,
```

```
[time.tempo_standard].[all].Children))))))
ON ROWS
from [traffic]
```

Query 6.

```
select {[Measures].[numerolog]}
ON COLUMNS,
Hierarchize
(Union(Crossjoin({[scn].[all]},
Crossjoin({[callendreason].[all]},
Union(Crossjoin({[disconnectcause].[all]},
{[time.tempo_standard].[all]}),
Crossjoin({[disconnectcause].[all]},
[time.tempo_standard].[all].Children)))),
Crossjoin([scn].[all].Children,
Crossjoin({[callendreason].[all]},
Union(Crossjoin({[disconnectcause].[all]},
{[time.tempo_standard].[all]}),
Crossjoin({[disconnectcause].[all]},
[time.tempo_standard].[all].Children))))))
ON ROWS
from [traffic]
```

5.2.2 Experimental Results

The execution of the queries described above had different execution times depending on the execution schema as reported in Figure 6 (each cell represents the execution time of the query in milliseconds).

	MySQL StarSchema	MySQL FlatSchema	MonetDb StarSchema	MonetDB FlatSchema
Q1	8730	770	915	306
Q2	14493	2776	2619	1634
Q3	7959	4812	1039	1015
Q4	6455	2346	1000	245
Q5	9089	11511	2064	1870
Q6	8663	5947	3303	1470

Figure 6: Query Execution Times.

The highest performance gap has been obtained when comparing MySQL Star Schema and MonetDB Flat Schema as reported in Figure 7.

The above mentioned difference is even more accentuated when considering the results shown in Figure 8.

The denormalization process, which causes MonetDB on a flatten schema to have better performance, does not lead to the same results when using MySQL. Indeed, we can observe that MySQL does not always take advantage of the denormalization performed. In Figure 9, it is easy to see that the MySQL execution times increase despite the denormalization on Query5.

On MonetDB, the flattening process guarantees the best experimental performances, this is due to

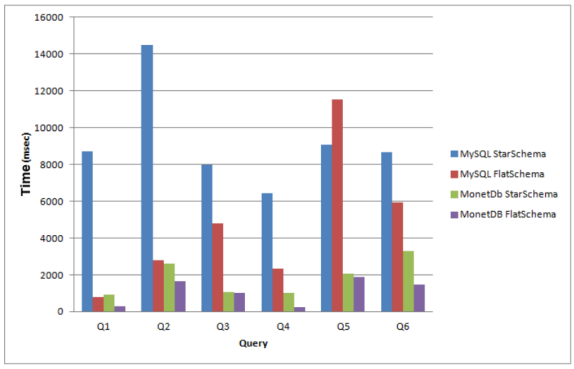

Figure 7: All Queries Result Times on different Schemes.

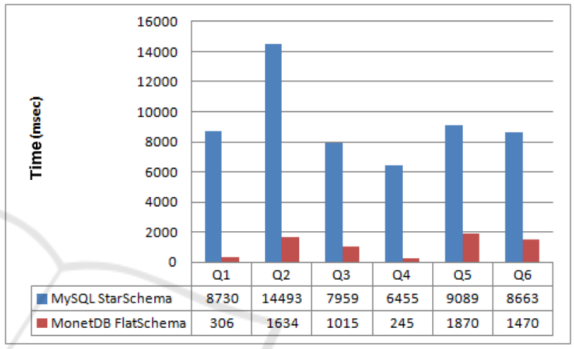

Figure 8: MySQL Star Schema vs MonetDB Flat Schema.

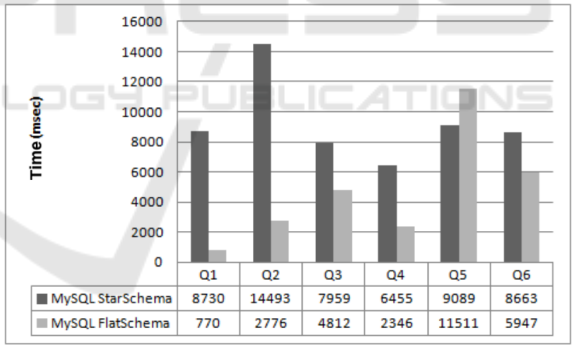

Figure 9: Comparison between MySQL Star Schema and MySQL Flat Schema.

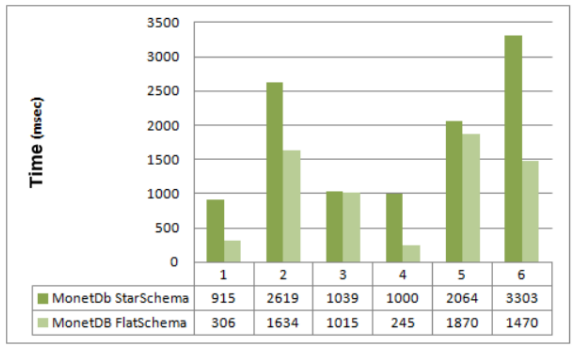

Figure 10: Comparison between MonetDB Star Schema and MonetDB Flat Schema.

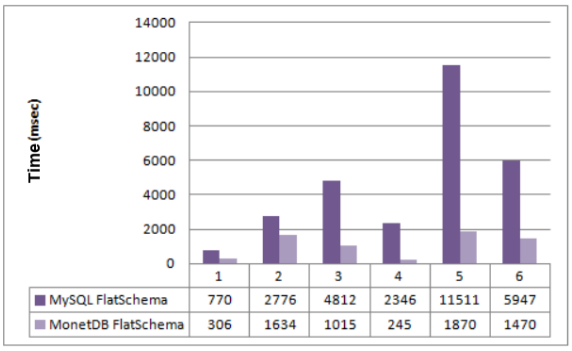

Figure 11: Comparison Flat Schema between MySQL and MonetDB.

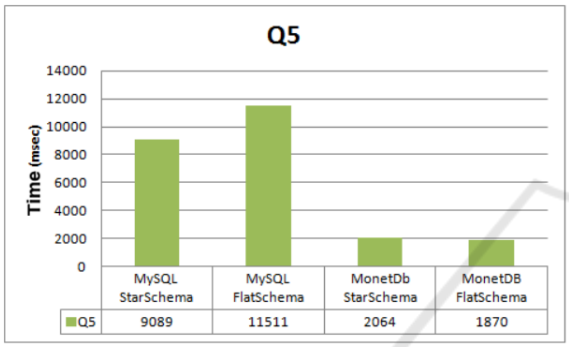

Figure 12: Query 5 an all scheme execution times.

MonetDB's internal operation, as can be seen from Figure 10. MonetDB with the same denormalized schema offers, compared to MySQL, better performance, as shown in Figure 11. This peculiarity is most relevant when comparing the results on the same query. Figure 12 takes into consideration the Query 5 which involves join operations with high cardinality.

6 CONCLUSION

Big data analysis is a challenging task as we need to take into account the velocity, variety and volume of information to be analyzed. Indeed, such features heavily influence the design of a system for big data warehousing. In this respect, we analyzed several design options in order to implement a prototype for Big Data Warehousing. Our MonOLAP prototype has been used for TETRA net analysis. Results on the efficiency of the system were quite satisfactory. We are now gathering real data from other public sources in order to perform a detailed analysis of the accuracy and effectiveness we can obtain by leveraging our prototype.

ACKNOWLEDGEMENTS

This work was supported by MIUR Cybersecurity project. We also thank Gaetano Fabiano for its contribution to system implementation.

REFERENCES

Agrawal et al., D. (2012). Challenges and opportunities with big data. A community white paper developed by leading researchers across the United States.

Economist (2010). Data, data everywhere. *The Economist*.

Economist (2011). Drowning in numbers - digital data will flood the planet - and help us understand it better. *The Economist*.

Labrinidis, A. and Jagadish, H. V. (2012). Challenges and opportunities with big data. *PVLDB*, 5(12):2032–2033.

Lohr, S. (2012). The age of big data. *nytimes.com*.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*.

Nature (2008). Big data. *Nature*.

Noguchi, Y. (2011a). Following digital breadcrumbs to big data gold. *National Public Radio*.

Noguchi, Y. (2011b). The search for analysts to make sense of big data. *National Public Radio*.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS