Step by Step Towards Energy-aware Data Warehouse Design

Ladjel Bellatreche[1], Amine Roukh[2], and Selma Bouarar[1]

[1] LIAS/ISAE-ENSMA - Poitiers University, France
(bellatreche, bouarars)@ensma.fr
[2] University of Mostaganem, Mostaganem, Algeria
roukh.amine@univ-mosta.dz

Abstract. Nowadays, the energy efficiency is one of the most challenging issues in the area of ITs. In data-centric applications, DBMSs are one of the major energy consumers. Reducing the energy consumption of *already deployed* data warehouses ($\mathcal{DW}$) and Eco-designing *new ones* represent a challenging issue. To deal with it, we can get inspired from the functional progresses done in others domains such as *Green Building*. The energy management in such domains is usually performed after making *audits* to evaluate different components of a building and the process of its construction life cycle. In this paper, we adapt these initiatives to the context of the $\mathcal{DW}$ design. We first elaborate two audits, one of the components of query optimizers that and takes the lion's share of energy consumption and another of the life cycle of $\mathcal{DW}$ design. Secondly, we propose a framework, supported by a tool (called *EnerQuery*) to satisfy two non-functional requirements: energy consumption and query performance. Thirdly, and thanks to the variability that has been widely studied by the community of software, we propose to vary the logical model of a given $\mathcal{DW}$ and evaluate its impact on optimizing queries. This variation is possible due to the relationships that may exist among the properties of a given $\mathcal{DW}$. Finally, intensive experiments are conducted using our energy cost models integrated in PostgreSQL query optimizers and an energy measurement tool to evaluate our finding on energy consumption at logical and physical levels.

1 Introduction

The COP21 (United nations conference on climate change)[3] event shows the willingness of countries (over 145 foreign Heads of State and Government attended the conference at Le Bourget, Paris, France), companies[4], individuals, government and non-government associations[5], etc. to save the planet. According to the 2009 Climate Action Plan[6], electricity is one of the two largest sources of greenhouse gas (GHG)

[3] http://www.gouvernement.fr/en/cop21
[4] http://climateaction.unfccc.int/assets/downloads/LPAA_-_Private_sector_engagement.pdf
[5] http://www.energyforall.info/
[6] http://greencomputing.berkeley.edu/

emissions for the campus and Information Technology (IT) is currently estimated to be responsible for approximately 10 percent of that electricity usage. IT has become a critical resource for the mission of the campus and usage of computing equipment continues to increase.

The continued expansion of the industry means that the energy uses by data centers, and the associated emissions of GHGs and other air pollutants, will continue to grow. Industry experts, such as the *SMARTer 2020*, reports that global data center emissions will grow 7 percent year-on-year through 2020 [23]. In a typical data center, *DBMS* is *one of the most important consumers of computational resources among other software deployed*, which turns DBMS to be a considerable energy consumer [48]. Reducing the energy consumption in the context of databases ($\mathcal{DB}$s) become a major issue. The *Beckman report* on databases published in February 2016, has already highlighted the importance of integrating the energy dimension in the process of designing databases [1].

Face to these requirements, the $\mathcal{DB}$ community did not stand idly, but from last decade, it continuously proposes initiatives around four main actions (that we call them **OBRE**: **O**ffer, **B**orrow, **R**eform, **E**valuate) to deal with energy.

- **Offer:** the database technology was made available for energy professional for the analysis usage to enable smarter scheduling of energy consumption of entities such as smart cities (in MIRABEL project [58]) and electrical vehicles in EDF (*Electricité De France*) project [55].
- **Borrow:** the $\mathcal{DB}$ technology employs green hardware and platforms to deploy the target database applications [30].
- **Reform:** the $\mathcal{DB}$ community did several efforts in reforming their software to integrate energy. These efforts concern mainly the development of cost models to estimate energy and then use them to (i) generate query plans [75,34,35] and to (ii) select optimization structures such as materialized views [53].
- **Evaluate:** to test their efficiency and effectiveness, energy initiatives have to be evaluated either using real datasets or benchmarks [53].

The set of tasks of *Reform* action assumes that the $\mathcal{DB}$ is already deployed in a given DBMS. To further reduce the energy, this action has to review different phases of the life cycle of the $\mathcal{DB}/\mathcal{DW}$ to make them potentially energy-sensitive.

Note there several efforts and initiatives to integrate energy have been launched in other domains such as private and public building. For instance, the *Australian government* has had an energy efficiency strategy in place since 2002[7]. This strategy describes the measures that have been taken by in terms of energy management within government buildings. These measures apply at *each stage in a building's life*:

- design and construction of new government buildings;
- refurbishment of existing government buildings;
- negotiation of new and renewed leases;
- ongoing building management and maintenance;
- equipment procurement.

Based on these experiences, we can easily make the parallel between the Green Building domain and the $\mathcal{DB}$ technology. As a consequence, *we claim that the energy* has to be

[7] `https://www.sa.gov.au/topics/property-and-land/land-and-property-development/building-rules-regulations-and-information/sustainability-and-efficiency-regulations/government-energy-efficiency-initiatives`

integrated in: (1) the query optimizers of DBMSs hosting $\mathcal{DB}/\mathcal{DW}$ and in (2) different phases of the life cycle of $\mathcal{DB}/\mathcal{DW}$ design.

A query optimizer represents one of the main components of DBMS in terms of energy consumption [75,34,35,53,54]. There has been extensive studies on the query optimization, since the early 70s. Several algorithms and systems have been proposed, such as *System-R project* [11], where its findings have been widely incorporated in many commercial query optimizers.

Advanced query optimizers perform two main tasks: **(i)** enumeration of execution plans of a given query and **(ii)** the selection of the best plan. This selection uses cost-based optimization (CBO). The CBO is a mathematical processor, where it uses formulas to calculate the cost of evaluating a plan. This cost is dedicated to estimate several non-functional requirements (NFRs) such as the number of inputs-outputs, CPU, communication, etc. A CBO driven-approach is suitable when statistics on tables, indexes, selectivity factors of join and selection predicates, etc. are available. A straightforward integration of the energy concerns its incorporation in the cost models of the query optimizers.

It should be noticed that in already deployed $\mathcal{DW}$, its logical model is frozen. This assumption surely becomes questionable when integrating energy in the design, since it ignores the chained aspect of the life cycle. Knowing that many variants of a logical schema may exist. This is due to the presence of dependencies and hierarchies among attributes of a given warehouse application; it is worth studying the impact of this variation on the energy consumption.

In this paper, we consider these two issues for energy saving when optimizing and constructing a $\mathcal{DW}$. For the first issue (considering already deployed databases), we elaborate an audit on DBMS components, in general, and the query optimizers, in particular that help us to identify energy-sensitive components that may significantly impact the energy. To design energy-aware query optimizers, we recommend a deep understanding of their functionalities. To satisfy this requirement, we use *PostgreSQL* DBMS, considered as the world's *most advanced open source database*. The examination of its query optimizer allows us making PostgreSQL energy sensitive. A demonstration of our tool is accepted in ACM CIKM'2016 [54][8].

To integrate the energy through all phases of the life cycle of the $\mathcal{DB}/\mathcal{DW}$ design, we consider a second audit for each phase. This audit allows us understanding different characteristics of each phase and the interdependencies that may exist between them. This audit contributes in identifying a strong dependency between logical and physical phases. This motivates us to incorporate the energy at the logical phase.

The paper is organized as follows: Section 2 presents a discussion about the existing works on energy based on the four actions that we have mentioned in the Introduction Section which are **O**ffer, **B**orrow, **R**eform and **E**valuate. Section 3 gives some definitions about energy. Section 4 presents our first contribution in integrating the energy in query optimizers for already deployed $\mathcal{DB}/\mathcal{DW}$. Section 5 presents in details the audit performed on different phases of the life cycle of $\mathcal{DB}/\mathcal{DW}$ design, and gives some hints to integrate energy at the logical level. Finally, Section 6 concludes the paper by summarizing the main results and suggesting future work.

2 Related Work

The most existing studies of our state-of-art are discussed our *OBRE* actions.

[8] `https://forge.lias-lab.fr/projects/ecoprod`

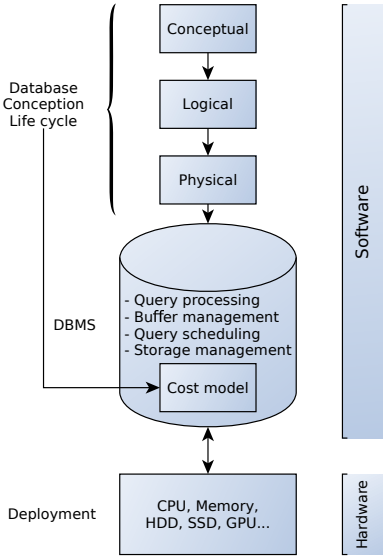


Fig. 1: Energy integration levels in DBMS

Offer. As we said in the Introduction, the database technology is used in the past and now to store energetic data from vehicles, smart cities, etc. The *MIRABEL Project* [58] is an example of this direction. It consists in developing an approach on a conceptual and an infrastructural level that allows energy distribution companies balancing the available supply of renewable energy sources and the current demand in ad-hoc fashion. It uses a DBMS to store forecasting in order to answer time series queries, which is an important functionality in energy data management [22,45].

Borrow. As other technology, databases never stop borrowing green hardware and platforms for their applications and DBMS [30].

Reform. This aspect consists in reforming existing software components to minimize their energy use. They concern mainly two aspects: (1) the definition of cost models to predict energy and (2) the proposition of optimization techniques to reduce energy.

– **Energy cost models** Prior works have been concentrated on building power cost models to predict query power consumption. In [75,74], the authors discussed the opportunities for energy-based query optimization, and a power cost model is developed in the conjunction of PostgreSQL's cost model to predict the query power consumption. A static power profile for each basic database operation in query processing is defined. The power cost of a plan can be calculated from the basic SQL operations, like CPU power cost to access tuple, power cost for reading/writing one page, and so on, via different access methods and join operations using regression techniques. The authors adapt their static model to dynamic workloads using a feedback control mechanism to periodically update model parameters using real-time energy measurements. The authors of [34] propose a technique for modeling the peak power of database operations. A pipeline-based model of query execution

plans was developed to identify the sources of the peak power consumption for a query and to recommend plans with low peak power. For each of these pipelines, a mathematical function is applied, which takes as input the rates and sizes of the data flowing through the pipeline operators, and as output an estimation of the peak power consumption. The authors used piece-wise regression technique to build their cost model. In the same direction, the work of [35] proposes a framework for energy-aware database query processing. It augments query plans produced by traditional query optimizer with an energy consumption prediction for some specific database operators like select, project and join using linear regression technique. [51] attempts to model energy and peak power of simple selection queries on single relations using linear regression. In our previous works [52], we proposed cost models to predict the power consumption of single and concurrent queries. Our model is based on pipeline segmenting of the query and predicting their power based on its Inputs-outputs (IO) and CPU costs, using polynomial regression techniques.

– **Optimization techniques** The presence of energy consumption cost models motivate the research community to propose cost-driven techniques. The work in [36] proposed an Improved Query Energy-Efficiency (QED) by *Introducing Explicit Delays mechanism*, which uses query aggregation to leverage common components of queries in a workload. The work of [35] showed that processing a query as fast as possible does not always turn out to be the most energy-efficient way to operate a DBMS. Based on their proposed framework, they choose query plans that reduce energy consumption. In [34], cost-based driven approach is proposed to generate query plans minimizing the peak power. In [53], genetic algorithm with a fitness function based on a energy consumption cost model, is given to select materialize views reducing energy and optimizing queries. The work by Xu *et al.* [74] is close in spirit to our proposal in this paper. They integrate their cost model into the DBMS to choose query plans with a low power at the optimization phase. However, they do not study the consumed energy at each phase of query optimizers. Moreover, they use a simple cost model that do not capture the relationship between the model parameters.

Evaluate. Energy evaluation is a sensitive point and as a consequence it requires accurate and *transparent* evaluation to show its savings. For transparent perspective, we propose an open platform available at the forge of our laboratory[9], allowing researchers, industrials and students to evaluate it.

3 Background

To facilitate the understanding of our proposal, some concepts and definitions are given related to energy and the life cycle of the data warehouse design.

Definition 1. *The energy is referred to as the ability to do work.*

Definition 2. *The power represents the rate of doing work, or energy per a unit of time.*

Energy is usually measured in joules while power is measured in watts. Formally, energy and power can be defined as:

$$P = \frac{W}{T} \tag{1}$$

[9] http://www.lias-lab.fr/forge/projects/ecoprod

$$E = P \times T \tag{2}$$

Where P, T, W, and E represent respectively, a power, a period, the total work performed in that period of time, and the energy.

The power consumption of a given system can be divided into two parts (i) baseline power and (ii) active power.

Definition 3. *The Baseline power is the power dissipation when the machine is idle. This includes the power consumption of the CPU, memory, I/O, fans, and other motherboard components in their idle state.*

Definition 4. *Active power is the power dissipation due to the execution of the workload. The active power component is determined by the kind of workload that executes on the machine and the way it utilizes CPU, memory, and I/O components.*

There exist two concepts of power that have to be considered during the evaluation of power utilization in DBMS. Average power representing the average power consumed during the query execution and peak power representing the maximum power. In this paper, we consider the average power.

The energy consumption can be reduced if either the average power consumption or the time intervals are reduced. Since optimizations in improving the performance of queries are widely studied, we focus on the power part of the Equation 2 in this work. The first step is to model the power[10] in order to estimate its consumption by the queries.

4 Energy-aware Query Optimizers

In order to build energy-aware query optimizers, we first propose an audit of their components to understand whether they may be energy-sensitive or not. After our audit, we present in details our framework for designing energy-aware query optimizers.

4.1 An Audit of Query Optimizers

Recall that a query optimizer is responsible for executing queries respecting one or several non-functional requirements such as response time. The process of executing a given query passes through four main steps: (i) parsing, (ii) rewriting, (iii) planning and optimizing and (iv) executing (cf. Figure 2). To illustrate these steps, we consider PostgreSQL DBMS as a case study.

4.2 Parse

The parser has to check the query string for valid syntax using a set of grammar rules. If the syntax is correct, a *parse tree* is built up and handed back. After the parser completes, the transformation process takes the parse tree as input and does the semantic interpretation needed to understand which tables, functions, and operators are referenced by the query. The data structure that is built to represent this information is called the *query tree*. The cost of this phase is generally ignored by the DBMS since it finishes very quickly. We follow the same logic and suppose that the energy consumption is negligent.

[10] For the rest of the paper, we will use interchangeably the terms *energy* and *power*.

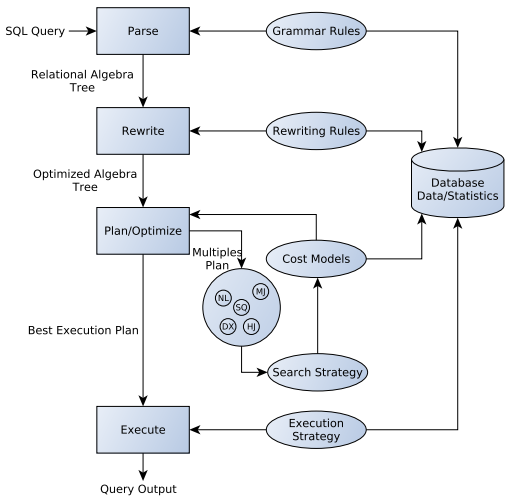


Fig. 2: Query optimizer steps

4.3 Rewrite

The query rewrite processes the tree handed back by the parser stage and it rewrites the tree to an alternate using a set of rules. The rules are system or user defined. This rules-based phase is also used in materialization views query rewriting. As for the previous step, the cost is ignored due to the fast completion.

4.4 Plan/Optimize

The task of the planner/optimizer is to create an optimal execution plan. A given SQL query can be actually executed in different ways, each of which will produce the same set of results. The optimizer's task is to estimate the cost of executing each plan using a cost-based approach and find out which one is expected to run the fastest.

Plan. The planner starts by generating plans for scanning each individual relation (table) used in the query. The possible plans are determined by the available *indexes* on each relation. There is always the possibility of performing a *sequential scan* on a relation, so a sequential scan plan is always created. If the query requires joining two or more relations, plans for joining relations are considered after all feasible plans have been found for scanning single relations. The available join strategies are: *nested loop join*, *merge join*, *hash join*. When the query involves more than two relations, the final result must be built up by a tree of join steps, each with two inputs. The planner examines different possible join sequences to find the cheapest one. If the query uses less than a certain defined threshold, a near-exhaustive search is conducted to find the best join sequences; otherwise, a heuristics based genetic algorithm is used.

To study the effects of such searching strategies, let us consider the query *Q8* of the TPC-H benchmark[11]. This a complex query which involves the join of 7 tables.

[11] http://www.tpc.org/tpch/

We modify the planner of PostgreSQL in three manners: (i) searching for a plan by employing actual DBMS strategy (ii) using the genetic algorithm, and (iii) manually by forcing the planner to choose a certain plan. For each strategy, we calculate its execution time, and the total energy consumption during query execution against 10GB datasets. Results are presented in Table 1.

Table 1: Planning step for TPC-H *Q8* with different searching strategies.

Search Algo	Planning Time (s)	Energy (j)
Default	0.110006	5200.362
GA	0.977013	5387.648
Manual	0.092054	5160.036

From the table, we can see that setting the query plan manually gives the betters results, in both time and energy. While the default searching algorithms (semi-exhaustive) leads to a slightly more execution time and energy consumption. The genetic algorithm gives the worst results in this example, perhaps due to the small number of tables in the query, since this strategy is used by the DBMS where there are more than 12 tables. Considering this small number of tables, if we go in real operational databases where there are a hundred of tables, the searching strategy used by the planner can lead to a noticeable energy consumption. Thus, setting the query plan of queries manually by the database administrator is recommended in large databases to gain in energy efficiency.

Optimize To evaluate the response time for each execution plan, cost functions are defined for each basic SQL operator. The general formula to estimate the cost of operator *op* can be expressed as:

$$Cost_{op} = \alpha \times I/O \ \oplus \ \beta \times CPU \ \oplus \ \gamma \times Net \tag{3}$$

Where I/O, CPU, and Net are the estimated pages numbers, tuples numbers, communication messages, respectively, required to execute *op*. They are usually calculated using database statistics and selectivity formulas. The coefficients α, β and γ are used to convert estimations to the desired unit (e.g, time, energy). $\oplus$ represents the relationship between the parameters (linear, non-linear). The coefficient parameters and their relationship can be obtained using various techniques such as calibration, regression, and statistics. Thus, an energy cost model must be defined at this stage with the relevant parameters.

The finished plan tree consists of sequential or index scans of the base relations, plus nested-loop, merge or hash join nodes as needed, plus any auxiliary steps, such as sort nodes or aggregate-function calculation nodes.

4.5 Executor

The executor takes the plan created by the planner/optimizer and recursively processes it to extract the required set of rows. This is essentially a demand-pull *pipeline*

mechanism. Each time a plan node is called, it must deliver one more row, or report that it is done delivering rows. Complex queries can involve many levels of plan nodes, but the general approach is the same: each node computes and returns its next output row each time it is called. Each node is also responsible for applying any selection or projection expressions that were assigned to it by the planner.

To study the effect of the execution step on designing green-query optimizer, we consider an example of query *Q22* from the TPC-H benchmark. Figure 3 presents the execution plan returned by the PostgreSQL query optimizer. As we showed in [52], the power consumption is directly influenced by execution model of the DBMS. Therefore, the execution plan can be divided into a set of segments, we refer to these segments as *pipelines*, where the pipelines are the concurrent execution of a contiguous sequence of operators. The pipeline segmentation of the optimizer plan for query *Q22* is shown in Figure 3, there are 4 pipelines, and a partial order of the execution of these pipelines is enforced by their terminal *blocking* operators (e.g, PL3 cannot begin until PL2 is complete).

In our previous study, we showed that *when a query switches from one pipeline to another, its power consumption also changes*. During the execution of a pipeline, the power consumption usually tends to be approximately constant [52]. Therefore, the pipelining execution is very important and has a direct impact on power consumption during query execution. The design of green query optimizer should take into consideration the execution strategy, which is unfortunately ignored by Xu *et al.* [74].

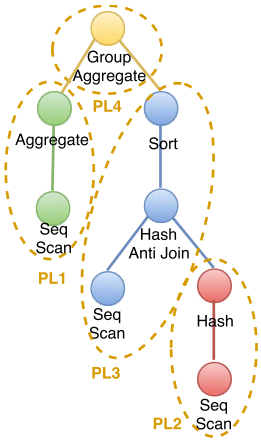


Fig. 3: Execution plan of TPC-H benchmark query *Q22* with corresponding pipeline annotation.

4.6 Our Energy-aware Query Optimizer

In this section, we describe the design and the implementation of our proposal into PostgreSQL database. As we mentioned above, the planner/optimizer and the executor stages have an impact on energy consumption and should be considered in designing

any green-query optimizer. We extended the cost model, the query optimizer and the communication interface of PostgreSQL to include the energy dimension.

Inspired by the observation made in the previous section, we designed our cost-based power model. The basic idea of this model is to decompose an execution plan into a set of power independent pipelines delimited by blocking/semi-blocking operators. Then for each pipeline, we estimate its power consumption based on its CPU and I/O cost.

The work-flow of our methodology is described in Figure 4.

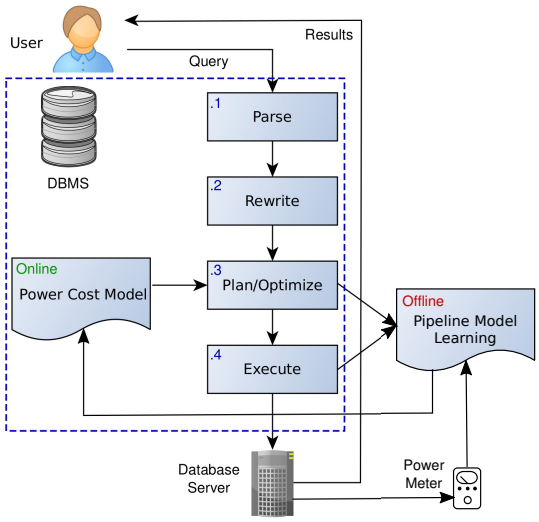


Fig. 4: The Design Methodology

4.7 Power Cost Model

In this section, we present our methodology for estimating energy consumption. The characteristics of our model include: (i) the segmentation of an execution plan into a set of pipelines, (ii) the utilization of the pipeline cost to build the regression model, and (iii) the estimation of the power of future pipeline based on pipeline cost and the regression equation.

Pipeline Segmentation. When a query is submitted to the DBMS, the query optimizer chooses an execution plan (cf. Figure 3). A physical operator can be either *blocking* or *nonblocking*. An operator is blocking if it cannot produce any output tuple without reading at least one of its inputs (e.g, sort operator). Based on the notion of blocking/nonblocking operators, we decompose a plan in a set of pipelines delimited by blocking operators. Thus, a pipeline consists of a set of concurrently running operators [21]. As in previous work [21], the pipelines are created in an inductive manner, starting

from the leaf operators of the plan. Whenever we encounter a blocking operator, the current pipeline ends, and a new pipeline starts. As a result, the original execution plan can be viewed as a tree of pipelines, as showed in Figure 3.

Model Parameters. Given a certain query, the query optimizer is responsible for estimating CPU and I/O costs. Our strategy for pipeline modeling is to extend the cost models that are built into the PostgreSQL DBMS for query optimization. To process a query, each operator in a pipeline needs to perform CPU and/or I/O tasks. The cost of these tasks represents the "cost of the pipeline", which is the active power to be consumed in order to finish the tasks. In this paper, we focus on a single server setup and leave the study of distributed databases as future work. Thus, the communication cost can be ignored. More formally, for a given query Q composed of p pipelines $\{PL_1, PL_2, \ldots, PL_p\}$. The power cost $Power(Q)$ of the query Q is given by the following equation:

$$Power(Q) = \frac{\sum_{i=1}^{p} Power(PL_i) * Time(PL_i)}{Time(Q)} \tag{4}$$

The *time* variable represents the pipelines and the query estimated time to finish the execution. Unlike Xu *et al.* study which ignores the execution time [75], in our model, the time is an important factor in determining the CPU or I/O dominated pipeline in a query. The DBMS statistics module provide us with this information. Let a pipeline PL_i composed of n algebraic operations $\{OP_1, OP_2, \ldots, OP_n\}$. The power cost $Power(PL_i)$ of the pipeline PL_i is the sum of CPU and I/O costs of all its operators:

$$Power(PL_i) = \beta_{cpu} \times \sum_{j=1}^{n_i} CPU\_COST_j + \beta_{io} \times \sum_{j=1}^{n_i} IO\_COST_j \tag{5}$$

where β_{cpu} and β_{io} are the model parameters (i.e., unit power costs) for the pipelines. For a given query, the optimizer uses the query plan, cardinality estimates, and cost equations for the operators in the plan to generate counts for various types of I/O and CPU operations. It then converts these counts to time by using system-specific parameters such as CPU speed and I/O transfer speed. Therefore, in our model, we take I/O and CPU estimations already available in PostgreSQL before converting it to time. The IO_COST is the predicted number of *I/O* it will require for DBMS to run the specified operator. The CPU_COST is the predicted number of *CPU Tuples* it will require for DBMS to run the specified operator.

Parameters Calibration The key challenge in equation (5) is to find model parameters β_{cpu} and β_{io}. Simple linear regression technique, as used in [75,34,35], did not work well in our experiments, especially when data size changes, this is because the relationships between data size and power are not linear. In other words, processing large files does not *always* translate in high power consumption. It depends more on the type of queries (I/O or CPU intensive) and their execution time. Therefore, we employed multiple polynomial regression techniques. This method is suitable when there is a *nonlinear* relationship between the independents variables and the corresponding dependent variable. Based on our experiments, the order *m=4* gives us the best results (the residual sum of squares is the smallest). The power cost $Power(PL_i)$ of the

pipeline PL_i is computed as:

$$\begin{aligned} Power(PL_i) = \beta_0 + \beta_1(IO\_COST) + \beta_2(CPU\_COST) + \\ \beta_3(IO\_COST^2) + \beta_4(CPU\_COST^2) + \\ \beta_5(IO\_COST \times CPU\_COST) + \\ \cdots + \\ \beta_{13}(IO\_COST^4) + \beta_{14}(CPU\_COST^4) + \epsilon \end{aligned} \tag{6}$$

Where IO_COST, CPU_COST denote the pipeline I/O and CPU costs respectively, these costs are calculated using the DBMS cost model functions, and ϵ is a noise term that can account for measurement error. The β parameters are regression coefficients that will be estimated while learning the model from training data. Thus, the regression models are solved by estimating the model parameters β, and this is typically done by finding the least-squares solution [42].

4.8 Plans Evaluation

The query optimizer evaluates each possible execution path and takes the fastest. Adding the energy criterion, we must adjust the comparison functions to reflect the tradeoffs between energy cost and processing time. In order to give the database administrator a solution with the desired trade-off, we propose to use the weighted sum of the cost functions method. In this scalarization method, we calculate the weighted sum of the cost functions so as to aggregate criterions and have an equivalent single criterion to be minimized. This method is defined as follows:

$$\begin{aligned} minimize \quad & y = f(x) = \sum_{i=1}^{k} \omega_i \cdot f_i(\overrightarrow{x}) \\ & \sum_{i=1}^{k} \omega_i = 1 \end{aligned} \tag{7}$$

Where ω_i are the weighting coefficients representing the relative importance of the k cost functions. $f_i(x)$ represents power cost function and performance cost function respectively. We implemented these two coefficients as an external parameter in the DBMS, so the database administrator or users can change them on the fly.

Figure 5 shows the optimal query plan returned by the modified query planner/optimizer for TPC-H query *Q3* and how it changes when user preferences vary. Initially, we used a performance only optimization goal, the total estimated processing cost is 371080 and the total estimated power is 153. Changing the goal to be only power, the processing cost increased to 626035 but the power falls down to 120. In the trade-off configuration, the processing cost is 377426 and the power is 134. In Figure **??**, the nested loop operator draws the high amount of power in the query (33 watts) but the plan is chosen by the optimizer because it is very fast. In Figure **??**, we realize that the merge join operator is the slowest in the query, its processing cost is 539200, with its power being minimal. The two hash join operators used in Figure **??** give a good trade-off, for a 1.7% of performance degradation, we get 12.4% of power saving.

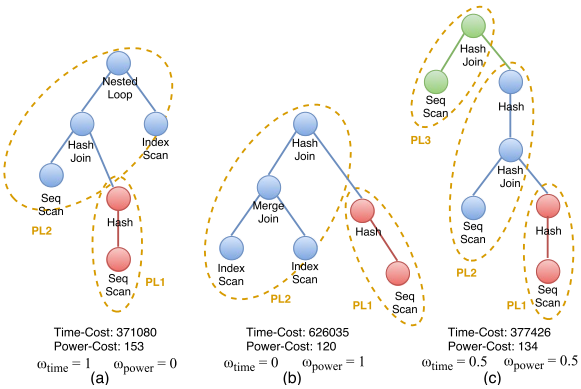


Fig. 5: The optimal plan for TPC-H query *Q3* when changing user preferences.

4.9 EnerQuery GUI

In this section, we describe the graphical user interface part of EnerQuery. The GUI helps to manipulate EnerQuery, changing parameters and showing in real time their impact on the power consumption.

The EnerQuery GUI interface is used to facilitate users manipulating the framework settings and seeing their effect on the system. The interface is implemented using C++ programming language and Qt library. Figure 6 gives an overview of the main GUI, which comprises several component modules:

Configuration. This module is responsible for the connection establishment with the DBMS server. Users can also specify the path for the power meter driver in order to capture real-time power consumption. The most important part here is the power/performance settings, which decide the optimization goals to be performance or power oriented.

SQL Query. In this module, users can give their SQL query to be executed. Queries supported vary from simple transactional operations to very complex reporting operations involving many tables with large data size. The execution is done in a separate thread and the results are displayed in a tree table widget.

Power Time-line. When the user executes a query, EnerQuery dynamically displays via the power meter the real-time power consumption. After the query finishes executing, the total energy that has been consumed during query execution time is computed and shown. This can give users a real observation of the energy that has been saved using the desired trade-off parameters. Also, users can compare the estimated and the real values to check the model accuracy or further refine it.

Execution Plan. When the user submits a query, the query optimizer will select their best execution plan with respect to the pre-defined trade-off. The execution plan

Fig. 6: EnerQuery main GUI and its component module panels.

is displayed with various information, such as estimated cost, power consumption, I/O and CPU costs for every physical operator through mouse-hovering events. Also, the pipeline notation is demonstrated, as shown in Figure 6 (4), the pipeline trees are grouped with the same color. The GUI shows how the trade-off parameters affect the generated plan. Thus, we can help users better understand and interpret runtime optimization information's and pipeline notation.

5 Audit of the $\mathcal{DW}$ Life Cycle

In this section, we present different phases of the life-cycle of its design inherited from traditional databases. Usually, it includes the following phases: **(a)** data source analysis, **(b)** elicitation of requirements, **(c)** conceptual, **(d)** logical, **(e)** deployment, and **(f)** physical design. ETL is appended as a design phase responsible for analytical processing [32] (Fig. 7).

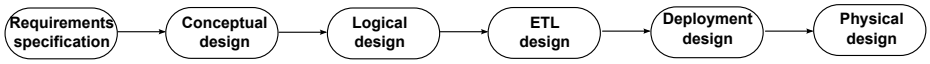


Fig. 7: Phases of $\mathcal{DW}$ design

5.1 Data Sources

As we said before a $\mathcal{DW}$ is built from *heterogeneous* data sources, with *different data representations*, and *provenance*. The heterogeneity poses serious problems when designing the $\mathcal{DW}$. This is due to the presence of conflicts that may exist among sources. Goh et al. [25] suggest the following taxonomy of conflicts: *naming conflicts*, *scaling conflicts*, *confounding conflicts* and *representation conflicts*. These conflicts may be encountered at *schema level* and *at data level*.

- *Naming conflicts:* occur when naming schemes of concepts differ significantly. The most frequently case is the presence of *synonyms* and *homonyms*. For instance, the status of a person means her familial status or her employment status.
- *Scaling conflicts*: occur when different reference systems are used to measure a value (for example *price* of a product can be given in Dollar or in Euro).
- *Confounding conflicts*: occur when concepts seem to have the same meaning, but differ in reality due to different measuring contexts. For example, the weight of a person depends on the date where it was measured. Among properties describing a data source, we can distinguish two types of properties: *context dependent properties* (e.g. the weight of a person) and *context non-dependent properties* (gender of a person).
- *Representation conflicts* : arise when two source schemas describe the same concept in different ways. For example, in one source, student's name is represented by two elements *FirstName and LastName* and in another one it is represented by only one element *Name*.

The data sources may be grouped into five main categories [49]: (i) *production data*, (ii) *internal data*, (iii) *archived data*, (iv) *external data* and (v) *experimental data*. The production data come from the various operational systems of the enterprise. The internal data include the data stored in spreadsheets, documents, customer profiles, databases which are not connected to the operational systems. External data such as data produced by external agencies, weather forecast services, social network, and knowledge bases such as Yago [29], etc. play an crucial role in $\mathcal{DW}$ design of adding new values to the warehouses [10]. Recently, the computational science community such as physics, aeronautic, etc. is building warehouses from the experiment results. We can cite for instance, the example of project AiiDA[12].
The data of sources range from traditional ones, to semantic data, passing by graph databases [28,24]).

5.2 $\mathcal{DW}$ Requirements

As any product, the development of a $\mathcal{DW}$ application is based on functional and non-functional requirements [27]. Note that functional requirements describe the functionalities, the functioning, and the usage of the $\mathcal{DW}$ applications to satisfy the goals and expectations of decision makers. These requirements are known as *business requirements* [19] that represent high-level objectives of the organization for the $\mathcal{DW}$ application. They identify the primary benefits that the $\mathcal{DW}$ technology brings to the organization and its users. They express business opportunities, business objectives and describe the typical users and organizations requirements and their added-values. Other functional requirements are associated to users of the $\mathcal{DW}$ (*called user requirements*) describe the tasks that the users must be able to accomplish with thanks to the $\mathcal{DW}$ application. User requirements must be collected from people who will actually use and work with this technology. Therefore, these users can describe both the tasks they need to perform with the $\mathcal{DW}$. These requirements are modelled using several formalisms such as: UML use cases, scenario descriptions and goals [15].

Non-functional requirements, called quality attributes are either optional requirements or needs/constraints [37], they are detailed in the system architecture. They describe how the system will do the following objectives: the security, the performance (e.g. response time, refresh time, processing time, data import/export, load time), the capacity (bandwidth transactions per hour, memory storage), the availability, the data integrity, the scalability, the energy, etc. This type of requirements has to be validated over the majority of the phases of the $\mathcal{DW}$ life-cycle.

5.3 Conceptual Design

This phase aims at deriving an implementation-independent and expressive conceptual schema according to the conceptual model. As said in [68] that database community agreed for several decades that conceptual models allow better communication between designers in terms of understanding application requirements. However, there is *no well-established* conceptual models for multidimensional data. Several formalisms mainly borrowed from traditional databases exist to design the conceptual model of a $\mathcal{DW}$: *E/R* model [26,56], object-oriented model [3,66,39], and no-graphical representation [46]. The quality of a $\mathcal{DW}$ conceptual model is evaluated using testing methods [64].

[12] `http://www.aiida.net/`

5.4 Logical Design

There are three main approaches to represent logical model of a $\mathcal{DW}$, depending on how the data cube is stored:

- *(i) the relational ROLAP* (ROLAP), which stores data cube in relational databases and uses an extension of SQL language to process these data. Two main schemes are offered by ROLAP: the star schema and snowflake schema. A star schema consists of a one or several large fact table (s) connected to multiple dimension tables via foreign keys. Dimension tables are relatively small compared to the fact table and rarely updated. They are typically non normalized so that the number of needed join operations are reduced. To avoids redundancy, the dimension tables of a star schema can be normalized. There is debate on the benefits of having such normalized dimension tables, since it will, in general, slow down query processing, but in some cases it provides a necessary logical separation of data such as in the case of the demographic information [38].
- *(ii) multidimensional OLAP (MOLAP)* stores cube data in multidimensional array format. The OLAP operations are easily implemented on this structure. For high dimensionality of data, ROLAP solutions are recommended [68]. (iii) *Hybrid OLAP (HOLAP)* combines both approaches. It gets benefit from the storage capacity of ROLAP and the processing power of MOLAP.

The $\mathcal{DW}$ is well known by the hierarchies and dependencies between properties [68].

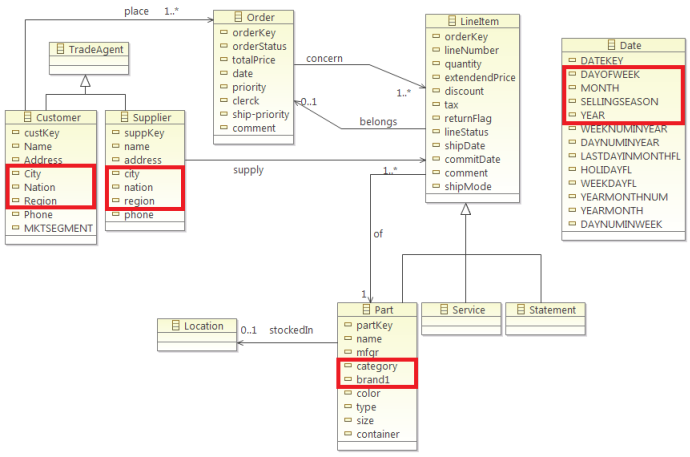


Fig. 8: $\mathcal{SSB}$ ontology

Fig.8 provides an overview of the global schema covering the domain of the Star Schema Benchmark $\mathcal{SSB}$, which is used further down for our experiments, and right below to illustrate the different types of existing correlations that we have identified and classified that way [13]:

- *Definition/equivalence relations or Generalization (DEF)*: when concepts/roles are defined in terms of other concepts/roles. E.g. a *Supplier* is a *TradeAgent* that *Supplies* a *LineItem*.
- *Inclusion dependencies or Specialization (ID)* Also called is-a relation or subsumption: it occurs when a concept/role is subsumed by another concept/role. E.g. *Customer* subsumes *TradeAgent*. When it concerns attributes, there is another application of this type: the notion of foreign keys, which states that the domain of one attribute must be a subset of the other correlated attribute.
- *Functional dependencies (CD/FD)*: we note CD if concepts and FD if attributes. They figure when a set of concepts/roles (or their instances) *determine* another set of the same type. E.g. *custKey* determines the *name* of *Customer*.
- *Multi-valued dependencies (MD)* or *soft dependencies*: specific to attributes, it is a generalization of the functional dependencies. Formally, the difference between the two is the abolition of the determination criterion, in other words, to a value set, we can associate more than one value set of the same type.
- *Conditional Integrity constraints (CIC)*: specific to attributes, they denote the $\mathcal{DB}$ integrity constraints (algebraic or semantic) [13] involving more than one attribute [50] and holding on instances of the relations. Note that definitions and dependencies are considered as simple integrity constraints which are valid on entire relations, contrary to conditional ones where the correlation is accompanied with condition(s) that must be respected. This latter aspect moves the application level from attribute range level to attribute values level. In other words, only a subset of the member attributes domain is involved (reduced range). E.g. *Customer.City=Paris→LineItem.discount*$\succ$ 20%. We distinguish two main categories: (i) conditional functional dependencies (CFD) [12] whereby the (FD) has to hold on a subset of the relation (a set of tuples) that satisfies a specific attribute pattern ([*Customer.city=London,Customer.name*]→ [*Customer.phone*]), rather than on the entire relation *Customer*, and (ii) more specifically, association rules that apply for particular values of some attributes [4] ($Part.color$='red'→ $Part.size$=50).
- *Hierarchies (H)*: specific to attributes, and more present in $\mathcal{DW}$s, where a set of attributes makes up a dimension hierarchy (e.g. Industry, category, product). They can be assimilated to the *part-whole* relationships. The particularity of this type, is that we could plan the construction of a new class for each hierarchy level.

From the former classification, we can infer the results achieved by exploiting these correlations throughout the design life-cycle of $\mathcal{DB}$. Those results belong to either conceptual, logical or physical levels. In more details:

- Conceptual level: the correlations having impact on the definition of conceptual schema are of type: DEF or ID. This impact consists of creating new concepts/roles (non canonical) when using DEF, or creating subsumption relationships linking the concepts/roles when using ID.
- Logical level: exploiting correlations of type CD, FD or H, has a direct impact on logical level: data normalization when using FD or CD, multidimensional $\mathcal{OLAP}$ annotation, hierarchical $\mathcal{OLAP}$ normalization when using H.
- Physical level: a lot of studies have exploited correlations of type MD, ID or CIC in the definition of the Physical Design Structures ($\mathcal{PDS}$).

In the light of the foregoing, we believe that any evolution/transition throughout the design life-cycle of $\mathcal{DB}$ can be controlled by correlations. Table 2 shows different studies in this field. In fact, thanks to the formal power of ontologies, and their strong similarity

[13] IC specify conditions/propositions that must be maintained as true ($Part.size$>0)

Studies \ Phases	MC	ML	MP	OLAP	Other
Anderlik & al. [6]				DEF/ID Roll-up	
Stohr & al. [63]		H Fragmentation			
Kimura & al. [33]			FD/MD $\mathcal{MV}$/indexes		
Brown & al. [18]			CIC Query optimizer		
Agrawal & al. [4]					CIC Data-mining
Petit & al. [47]	ID/FD ER schema	ID/FD Relational schema			ID/FD Reverse engineering

Table 2: Related work on correlations exploitation over the design life-cycle of $\mathcal{DW}$

with conceptual models, we can store correlations (identified by the $\mathcal{DB}$ users notably the designer) right from the conceptual phase. Afterwards, the transition to the logical level is henceforth based on correlations: namely the dependencies (CD, FD) for $\mathcal{DB}$, and hierarchies for $\mathcal{DW}$, as for the transition to the physical, it becomes controlled by either MD, ID or CIC. Indeed, several studies have shown that $\mathcal{DB}$ performance can be vastly improved by using $\mathcal{PDS}$ defined upon correlations, and even more when exploiting the interaction - generated upon correlations - between these $\mathcal{PDS}$, as is the case concerning $\mathcal{MV}$ and indexes in the *CORADD (for CORrelation Aware Database Designer for Materialized Views & Indexes) system* [33] (see Table 2).

5.5 ETL design

ETL processes are responsible for extracting and transforming data from heterogeneous business information sources to finally loading them into the target warehouse. This phase plays a crucial role in the process of the $\mathcal{DW}$ design. The quality of the warehouse strongly depends on ETL (the garbage in garbage out principle) [2]. The different steps of ETL need to understand different schemes (conceptual, logical and physical) of data sources [60,8,69,72,65,2,73,67]. Several commercial and academic tools of ETL are also available such as: Oracle Warehouse Builder (OWB), SAP Data Services, Talend Studio for Data Integration, Pentaho Data Integration, etc.[14].

The first studies on ETL were concentrated on the physical models of data sources that includes the deployment platform (centralized, parallel, etc.) and the storage models used by the physical models (e.g. tables, files). In [67], a set of algorithms was proposed to optimize the physical ETL design. Alkis et al. [60] propose algorithms for optimizing the efficiency and performance of ETL process. Other non-functional requirements such as freshness, recoverability, and reliability have been also considered [61]. The work of [43] proposes an automated data generation algorithm assuming the existing physical models for ETL to deal with the problem of data growing.

[14] `https://www.etltool.com/list-of-etl-tools/`

Other initiatives attempted to move backward ETL from physical models to logical models. [72] proposed an ETL work-flow modelled as a graph, where nodes of the graph are activities, record-sets, attributes, and the edges are the relationship between nodes defining ETL transformations. It should be noticed that graphs contributing in modelling ETL activities. In [69], a formal logical ETL model is given using Logical Data Language (LDL) [57] as a formal language for expressing the operational semantics of ETL activities. Such works assume that ETL workflow knows the logical models of data sources and the elements of their schemes (relations, attributes).

Another move backward to conceptual models has been also identified, where approaches based on ad-hoc formalisms [70], standard languages using UML [65], model driven architecture (MDA) [31], Business Process Model and Notation(BPMN) [73,5] and mapping modelling [20,40]. However, these approaches are based on semi-formal formalisms and do not allow the representation of semantic operations. They are only concentrated on the graphical design of ETL processes without specifying the operations and transformations needed to overcome the arising structural and semantic conflicts. Some works use ontologies as external resources to facilitate and automate the conceptual design of ETL process. [62] automated the ETL process by constructing an OWL ontology linking schemes of semi-structured and structured (relational) sources to a target data warehouse ($\mathcal{DW}$) schema. Other studies have been concentrated on semantic models of data sources such as the work of [44] that considers data source provided by the semantic Web and annotated by OWL ontologies. However, the ETL process in this work is dependent on the storage model used for instances which is the triples. Note that after each research progress on ETL, its operations have to be specified and rewritten. This situation motivates researchers to make ETL more generic. In [72], the authors propose a generic model of ETL activities that plays the role of a pivot model, where it can define 10 generic ETL operator [62]. The signature of each operator is personalized to the context where target data source contains integrated record-sets related to attributes extracted from sources satisfying constraints:

1. $Retrieve(S, A, R)$: retrieves record-sets R related to attributes A from Source S;
2. $Extract(S, A, R, CS)$: enables selection and extraction of data from source S satisfying constraint CS;
3. $Merge(S, A_1, A_2, R_1, R_2)$: merges record-sets R_1 and R_2 belonging to the same source S;
4. $Union(S_1, S_2, A_1, A_2, R_1, R_2)$: unifies record-sets R_1 and R_2 belonging to different sources S_1 and S_2 respectively;
5. $Filter(S, A, R, CS)$: filters incoming record-sets R, allowing only records with values satisfying constraints CS;
6. $Join(S, A_1, A_2, R_1, R_2)$: joins record-sets R_1 and R_2 having common attributes;
7. $Convert(S, A, R, F_S, F_T)$: converts incoming record-sets R from the format F_S of source S to the format of the target data source F_T;
8. $Aggregate(S, A, R, F)$: aggregates incoming record-set R applying the aggregation function F (count, sum, avg, max) defined in the target source.
9. $DD(R)$: detects and deletes duplicate values on the incoming record-sets R;
10. $Store(T, A, R)$: loads record-sets R related to attributes A in target data source T,

Building the ETL process is potentially one of the biggest tasks of building a warehouse; it is complex, time consuming. This is because it requires several tasks and

competencies in terms of modelling, work-flows, and implementations. Non-functional requirements such as quality and the performance are taken into account when designing ETL [71,59,17].

5.6 Deployment Phase

This phase consists in choosing the adequate platform in which the target warehouse will be deployed. Several platforms can candidates to store the warehouse: centralized, database clusters, parallel machines, Cloud, etc. The choice of the deployment platform depends on the company budget and the fixed non-functional requirements [9].

5.7 Physical design

It is a crucial phase of the $\mathcal{DW}$ life cycle. Note that the majority of non-functional requirements are evaluated during this phase. It uses the inputs of deployment and the logical phases. In this design, optimization structures such as materialized views, indexes, data partitioning, etc. are selected to optimize one or several non-functional requirements such as query performance and energy.

5.8 Summary

Based on the above discussion, we figure out that the energy may be included in all phases, and especial logical and physical ones. Recall that the physical phase in the funnel of all phases. This is characterized by the entries of its corresponding problem, called the physical design problem ($\mathcal{PDP}$). These entries are:

1. the chosen logical model ($\mathcal{LM}$) of the $\mathcal{DW}$ application identified during the logical phase of the life cycle;
2. the used query language ($\mathcal{QL}$) offered by the DBMS;
3. the used deployment platform ($\mathcal{DP}$) including the target DBMS and hardware;
4. the available logical and physical optimizations ($\mathcal{PO}$) offered by the DBMS that have to be exploited by DBA either in isolated or joint manners to ensure the performance of her/his workload. The logical optimizations refer to those offered by query optimizers (see Section 4). Whereas, the physical optimizations refer to the optimization structures ($\mathcal{OS}$) (e.g. materialized views, indexes, etc.) selected during the physical design phase.

The formalization of the $\mathcal{PDP}$ by taking into account the two non-functional requirements representing the query processing and the energy consumption is as follows: Given:

- A *workload* $\mathcal{W}$ of numerous queries, expressed in:
- a *query language* ($\mathcal{QL}$) related to:
- a *logical model* ($\mathcal{LM}$) translated to:
- a *physical model* ($\mathcal{PM}$) associated to:
- a set of *physical optimizations* ($\mathcal{PO}$) deployed in:
- a *platform* ($\mathcal{P}$).

$\mathcal{PDP}$ aims at picking optimization structures satisfying both requirements and respecting the given constraints (e.g., the storage). The resolution of our problem has to exploit as much as possible the dependencies between inputs and their variabilities. Fig. 9 presents an UML model illustrating our proposal.

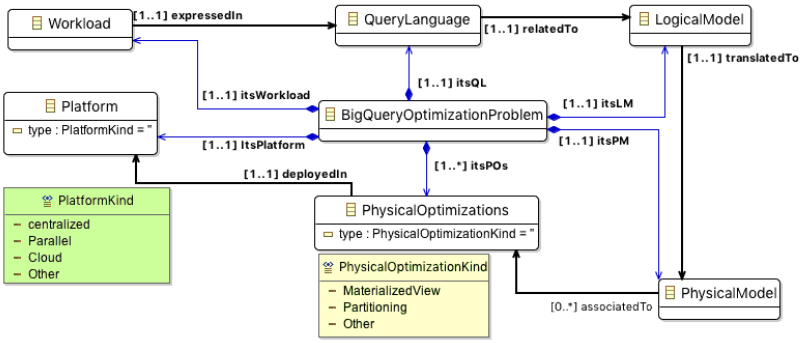


Fig. 9: An UML Model of our Big-Queries Optimization Problem

5.9 Energy at Logical and Physical Phases

To integrate the energy when logical designing a $\mathcal{DW}$, we can exploit the relationships that may exist between its properties. An analogy can be immediately drawn between the theory of the variability management ($\mathcal{VM}$) [7] (which is widely studied by the community of Software Engineering) and $\mathcal{PDP}$. In fact, this latter owns different variation points (dimensions, entries), as depicted in Fig. 10. Each variation point is seen as a complex search problem, often using a mathematical non-functional requirement-driven cost model to evaluate and select "best" solutions.

Based on this Figure, we can easily identify the *dependencies* among these dimensions. Actually, varying the logical schema strongly impacts the following entries: *workload*, the optimization structures ($\mathcal{OS}$) and the *constrains*.

Nevertheless, designers still intuitively fix a logical solution out of the large panoply of schemes, hence omitting eventual more relevant alternatives ($\mathcal{VM}$ aspect). Bearing this in mind, we fix some objectives to handle this missing piece in $\mathcal{PDP}$ puzzle: **(i)** capturing variability, **(ii)** *studying the impact* of variability on dependent components, as well as *efforts in terms of modeling and coding* to be spent by designers to manage this variability, and **(iii)** validating our methodology.

To show the impact of the variability of the logical phase on the physical model, we consider the following entries of our $\mathcal{PDP}$: *(a)* a Star Schema Benchmark (SSB)

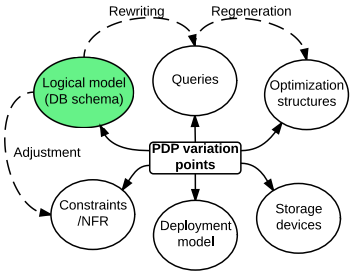


Fig. 10: Variation points of physical design problem.

as the $\mathcal{DW}$ logical schema, *(b)* query performance and energy consumption as non-functional requirements, *(c)* SSB workload, *(d)* materialized views as a $\mathcal{OS}$ and logical optimizations offered by the target DBMS, *(e)* the storage constraint dedicated to this $\mathcal{OS}$.

To the best of our knowledge, our proposal is the sole that studies the variation of the logical schema and evaluates its impact on the $\mathcal{PDP}$ dimensions, according to both energy consumption and query performance [14].

As we said before, our proposal considers the majority of dimensions of variability axes. To show the real impact on the physical design and to reduce the complexity of treating all dimensions at once, we incrementally integrate dimensions. We start by evaluating the impact of the logical schema variation on the physical design when executing a workload, without considering optimization structures. Only optimizations offered by target DBMS (referred to as *logical optimization*) are considered. Secondly, we evaluate the impact of the logical variation on the problem of materialized view selection.

5.10 Scenario 1: Impact of $\mathcal{VM}$ on Logical Optimizations

To study the impact of variability of the logical model on the physical phase, we consider a *naive scenario*, where logical optimizations (e.g. join implementations, join ordering, etc.) are delegated to the query optimizer of the target DBMS (Oracle 11gR2 in our case) and advanced optimizations structures such as materialized views are absent. We consider the query performance and energy consumption as two *objective functions* when executing the workload. In practice, for each variant of the initial logical schema of our $\mathcal{DW}$, we compute both metrics. Note that each variant requires rewriting efforts of the initial workload. Algorithm 1 gives an overview of our approach:

Algorithm 1: Algorithm dedicated to Scenario 1.

Input: $\mathcal{DW}$ logical model: $\mathcal{DW} = \{F, D_1, D_2, ..., D_n\}$; $\mathcal{Q} = \{q_1, q_2, ..., q_m\}$;
Output: $\mathcal{DW}\prime$: $\mathcal{DW}$ logical schema having the *most suitable* performance/power-saving trade-off

```
Generate the different possible logical schemes;
for each generated schema do
    Calculate the size of the schema;
    for each query in the workload do
        Rewrite the query conforming to the target schema;
        Execute the query;
        Record the overall query power & its execution time;
    Calculate the time and power averages of queries;
Normalize power and time values;
Weight both objectives (power & time);
DW' = Schema having the minimum of the weighted sum;
```

Our algorithm provides us both metrics corresponding to query performance and energy consumption. In order to help designers choose the schema that best fits their

requirements, we initially propose to use the weighted sum of the objective functions method that allows formulating the desired trade-off between target non functional requirements. In this scalarization method, we calculate the weighted sum of the normalized objective functions to aggregate objectives and have an equivalent single objective function to be optimized. This method is defined as follows [76]:

$$minimize \quad y = f(x) = \sum_{i=1}^{k} \omega_i \cdot f_i(\overrightarrow{x}) / \sum_{i=1}^{k} \omega_i = 1 \tag{8}$$

Where ω_i are the weighting coefficients representing the relative importance of the k objective functions of the problem. For example, an eco-performing schema would have an $\omega_{pow} = \omega_{perf} = 0.5$) while a performance-oriented schema would have $\omega_{perf} > \omega_{pow}$, contrary to an eco-oriented schema ($\omega_{pow} > \omega_{perf}$). This technique is well suited when the *Pareto* front is convex, which is the case with our curve, as further illustrated in section 5.12.

5.11 Scenario 2: Impact of $\mathcal{VM}$ on Physical Optimizations

In this scenario, we leverage the previous one by considering an optimization structure representing materialized views [41]. In our study, we do not delegate the selection of materialized views to advisors of commercial DBMSs, we propose instead an algorithm selecting them according to our previous metrics. This selection is proven to be NP-hard problem, and has been subject to many studies [41]. The process of selecting views requires three main components [16]:

a) A data structure to capture the interaction among queries, like the *And-Or view graph* or *Multi-View Processing Plan* [41]. It puts the algebraic operations of queries all together in a certain order as an acyclic graph. Starting from base tables as leaf nodes to queries results as root nodes, through intermediate nodes: unary operations (like selection/projection) and binary ones (like join/union). Getting the optimal order between intermediate nodes –join ones in particular- determines the efficiency of the structure.
b) Algorithms (e.g. deterministic algorithms, randomized algorithms, etc. [16]) exploiting a such structure to pick the best configuration of materialized views.
c) Cost models estimating different non functional requirements.

a) The construction of the data structure Our proposal has the ability to consider very large number of queries. This is due to the data structure borrowed from hypergraphs [16] representing the global plan of the workload as well as the interaction among queries. The main steps of the process of selecting views are[16]:

– **Step 1:** Parse query workload to extract the different algebraic operations;
– **Step 2:** Construct the hypergraph He out of join nodes, such that every He represents a query and thus contains its different join nodes modeled as vertices;
– **Step 3:** Partition He into a set of connected components He_{sub} (disjoint sub-hypergraphs), to put interacting nodes together;
– **Step 4:** Order the nodes of each He_{sub} according to a benefit function that determines a pivot node at each pass;
– **Step 5:** Merge the resulting He_{sub} to generate the global structure.

Our contribution concerns the He construction, and the ordering of He_{sub} nodes (2nd & 4th steps). In fact, only star schemes are handled by the baseline approach[16],

unlike ours that considers any multidimensional model (star, snowflake, constellation). The difference lies in that there henceforth exists some extra-join nodes not involving any-more the fact table (joins between dimensions and sub-dimensions, which we have called *extra-joins* against *fact-joins*), and this leads to a totally different situation. As depicted in Fig. 11, the 2nd and the 4th arrangements are impossible configurations in the baseline approach and frequent ones in ours. Indeed, we will have more than one starting nodes in one connected component.

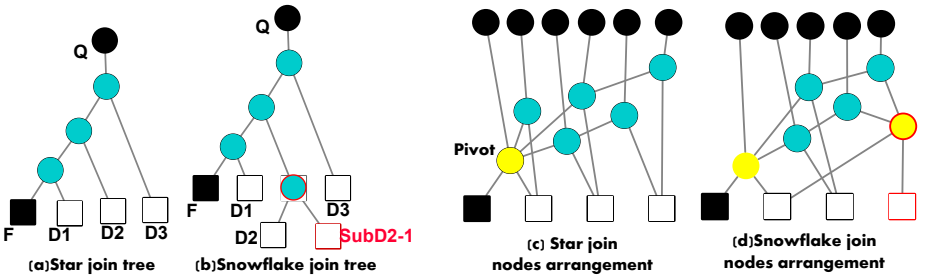


Fig. 11: Possible join nodes arrangement in a global plan.

However, the extra-joins introduce a partial order since they have to precede the fact-ones (sub-dimensions must be joined before their dimensions). This order must be considered when ordering nodes of He_{sub} (step4) so that: (i) the mother dimension must always appear after its extra-joins, (ii) the outer extra joins must always figure before the inner ones. These rules are guaranteed thanks to the below benefit functions, the purpose of which, is to find the pivot nodes and thus order the He_{sub} nodes.

Given: n_{f_i} a fact-join, n_{e_i} an extra-join, k the number of extra joins implied by a fact-join, *nbr* the number of queries using the node in question, *cost* its processing cost ⇒

$$\begin{cases} cost_{total}(n_{f_i}) = cost(n_{f_i}) + \sum_{j=1}^{k} cost(n_{e_j}) \\ benefit(n_{e_i}) = (nbr-1) * cost(n_{e_i}) - cost(n_{e_i}) \\ benefit(n_{f_i}) = (nbr-1) * cost(n_{f_i}) + \\ \qquad \sum_{j=1}^{k} (cost(nbr-1) * cost(n_{e_j})) \\ \qquad -cost(n_{f_i}) - \sum_{j=1}^{k} cost(n_{e_j}) \end{cases}$$

b) Materializing nodes and schema selection (Algorithm) Our approach, as summarized in algorithm2, is based on the *hyper-graph based* structure. In fact, if designer looks primarily for optimizing query performance, we will create this *structure* for each schema among the *top-k* performance-oriented schemes, materialize the pivot node of each one (the most advantageous node), execute queries for each schema and finally compare results. The schema having the smallest execution time of its queries will be the selected one. Otherwise, if designer needs to optimize both query performance and energy saving, the *structures* of the *top-k* eco-oriented schemes (or trade-off-oriented according to designer needs) will be generated.

Materializing the pivot node does not make sense anymore for saving power (because it is performance-oriented), nor all the join nodes because this would entail the

highest power consumption [53]. Testing 2^n possible configurations, where n is the number of join nodes, to find *Pareto* solutions, is impossible especially in DW workloads involving a lot of joins. A Pareto solution is a set of nodes (a view configuration), that would give - when materialized - values that can not be improved without making at least power or performance worse off. They are the best alternative since there does not typically exist a solution that minimizes all objective functions at once. Evolutionary Algorithms (EAs) are indeed suitable for multi-objectives optimization problems where large search spaces can be handled and multiple alternative trade-offs can be generated in a single optimization run [76]. The general idea behind an EA is to investigate a set of solutions that represent the Pareto optimal set as well as possible.

Algorithm 2: Algorithm dedicated to Scenario 2.

Input: NFR, $\mathcal{LM}$: a set of logical schemes / $\mathcal{LM}_i = \{F, D_j, SubD_{jk}\}/j \in \{1..n\}, k \in \{1..n_k\}, \mathcal{Q}_i = \{q_{i_1}, q_{i_2}, ..., q_{i_m}\}$[15]
Output: A set of join nodes to be materialized (view configuration)
for $\mathcal{LM}_i \in \mathcal{LM}$ **do**
  Generate the Multi View Processing Plan corresponding to its queries $\mathcal{Q}_i$;
  if *NFR = performance* **then**
    Each pivot node of each connected component of the structure is materialized;
  else `/* NFR = energy & performance */`
    Annotate each join node by its execution time and power consumption;
    Apply an evolutionary algorithm to select candidate views to be materialized optimizing *performance* as well as *energy*;
    Apply the weigthed sum on these candidates to select one view configuration;

c) Energy cost-model To evaluate the interest of the presence of a materialized view, without deploying, each time, the $\mathcal{DW}$ (schema and optimization structures), we adjust our mathematical cost model developed in [53]. In fact, this cost model has been constructed by assuming a DW with star schema. As a consequence, our adaptation consists in making it more generic to consider all variants of the logical schemes. This adaption mainly concerns the training phase that allows identifying the relevant parameters of our cost models using *polynomial* multivariate regression model [53].

5.12 Experimental Study

To evaluate the logical variability impact on physical design, we conduct intensive experiments related to our two scenarios. First, we present our development environment including hardware, software, datasets, and results.

[15] $\mathcal{LM}$ and $\mathcal{Q}$ are generated from Algorithm1.

Hardware setup Our machine is equipped with a "Watts UP? Pro ES[16]" power meter with one second as a maximum resolution. As commonly set up, the device is directly placed between the power supply and the $\mathcal{DB}$ workstation under test to measure the workstation's overall power consumption. The power values are logged and processed in a separate monitor machine (client-server architecture). We used a Dell PowerEdge R210 II workstation having an Intel Xeon E3-1230 V2 3.30 GHz processor, 10GB of DDR3 memory and a 2x500GB hard drive.

Software setup Our workstation machine is installed with the latest version of Oracle 11gR2 DBMS under Ubuntu 14.04 LTS with kernel 3.13 to minimize spurious influences, with 8192 as block size. We also disable unnecessary background tasks, clear the system and oracle cache for each query execution. We also disable unnecessary background tasks, clear the system and Oracle DBMS cache for each query execution.

Datasets We use SSB datasets with a scale factor of 10. It illustrates decision support systems that examine large volumes of data, and execute different types of queries with a high degree of complexity. We have identified the main hierarchies for each dimension table of the SSB multi-dimensional model, applied our formula ($H(Customer) * H(Part) * H(Supplier) * H(Date) = 2^{3-1} * 2^{2-1} * 2^{3-1} * 2^{4-1}$), and generated the resulting 256 possible schemes thanks to attributes correlations. As for workload, we create 30 queries based on SSB datasets, in such a way that two main categories must always be handled: (i) queries with operations that exhaust the system processor (CPU intensive queries) and (ii) queries with exhaustive storage subsystem resource operations (I/O intensive queries). Note that the considered queries include: queries with single table scan, others with multiple joins with different predicates. They also contain sorting/grouping conditions and simple and advanced aggregation functions. These queries are rewritten according to every schema [13].

Evaluation of Scenario 1 As already mentioned, the scenario 1 involved logical optimizations. In our case, we use the default optimizations offered by Oracle 11gR2 DBMS query optimizer. To conduct our experiments, we have deployed the different 256 schemes obtained from varying the initial SSB schema of our $\mathcal{DW}$. The initial queries of our workload are rewritten for each schema (7680 queries all in all) and executed. Execution time (from oracle) and power consumption (from power meter) are recorded. We first analyze one objective function "*power*", depicted in Fig. 12 that confirms power variation according to logical schema and, even better, shows that star schema is far from being the most eco-model. We have noticed that the *co-normalization of the smallest dimension tables* (supplier (2000*SF) and dates (2556) in this case) in the presence of CPU-intensive operations clearly disadvantages power consumption, but neither the number of joins (IO costs) nor the number of CPU-intensive operations (e.g. aggregations/sorting) influence directly the power consumption. A possible explanation is that most of query time execution is spent in CPU processing because data is read quickly due to the files small size. On the opposite, when most of query time execution is spent in waiting until data is ready because of data swapping between memory/disk, less power consumption is recorded.

In a second place, we consider two objective functions representing query performance and power consumption. We then highlight the relation between them which

[16] https://www.wattsupmeters.com/

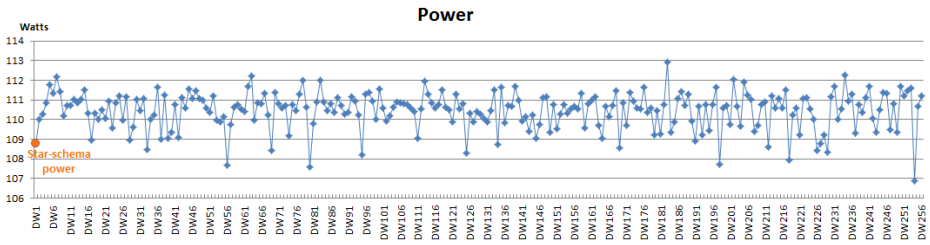


Fig. 12: Impact of logical design on $\mathcal{DW}$ power consumption.

takes the form of a convex as illustrated in Fig. 13. This reveals the existence of logical schemes optimizing both NFR (Pareto solutions), and meanwhile, approves our choice of weighting method in selecting schemes.

On the other hand, normalization process reduces storage space, especially with large dimension tables and/or important size of hierarchies. Snowflake schemes are hence appropriate for space-constrained applications as depicted in Fig. 14. This storage gain could be also propagated to storage constraints of optimization structures. These experiments show the limitations of the initial SSB schema to satisfy our fixed NFRs.

Evaluation of Scenario 2 The previous experiments took almost 10 days (7680 queries) what reveals the necessity of using a simulator (cost model) for these and future experiments. We focus in this scenario, on the problem of selecting materialized views by considering the variation of the logical schema (256 schemes), unlike current studies dealing with only one schema. To generalize this, we develop a Java-simulator tool that generates the global plan, using our hypergraph-based approach, for a given workload following any $\mathcal{DW}$ logical schema, and assessed the NFRs cost for the different schemes/workloads using pluggable cost models. Our simulator is equipped with *mathematical cost models estimating different metrics* (query performance, power consumption, etc.) [52]. Fig. 15 presents our simulation results of assessing performance (I/O) of the different workloads/schemes with/without views. This attests to the relevance of: (i) materializing views to query performance, which is quite expected. This

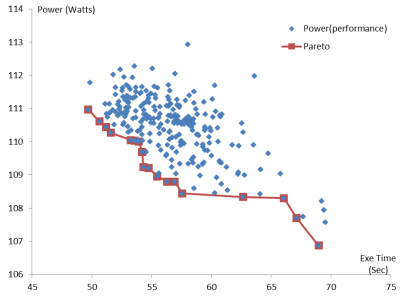


Fig. 13: Impact of logical design on $\mathcal{DW}$ power and performance.

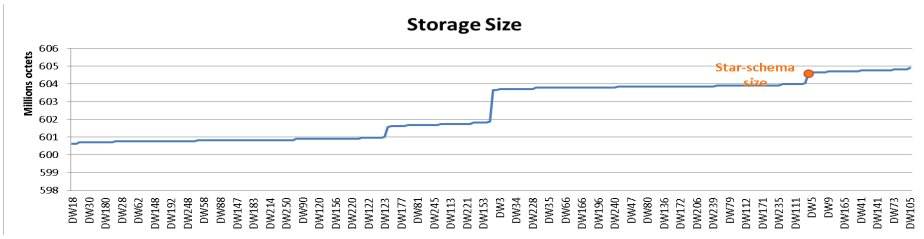


Fig. 14: Impact of logical design on $\mathcal{DW}$ size.

partially proves the coherence of our cost model, (ii) the impact of logical variability on physical design.

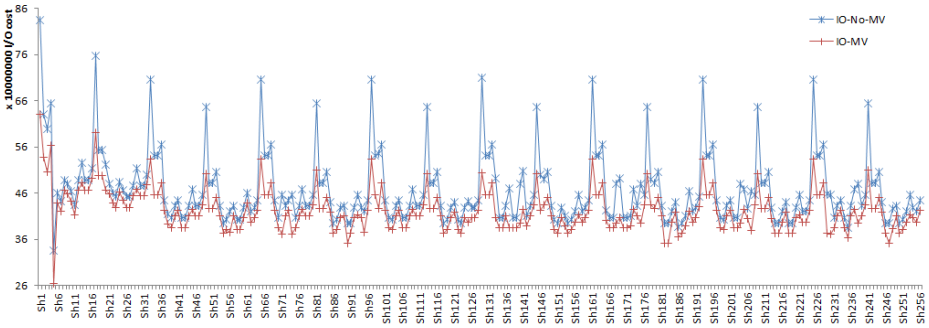
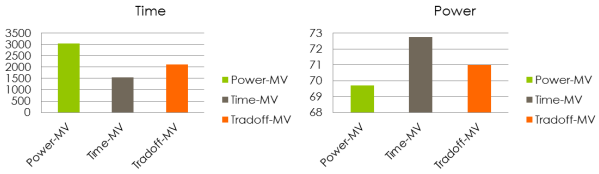


Fig. 15: Impact of $\mathcal{VM}$ on Performance of Physical Optimizations.

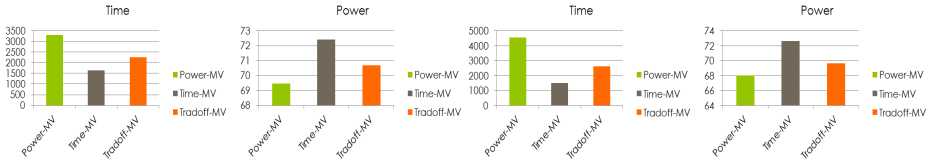
Impact of $\mathcal{VM}$ on both Power and Performance of Physical Optimizations. Rather than testing in combinatorial fashion all the configurations (256 schemes, 30 queries and n views generated by evolutionary algorithms, for each schema), it would make more sense to first select three different logical schemes from our first scenario (that can be hence done using our simulator): a performance, power and trade-off oriented schemes. Using *MOEA* [17] integrated to our tool, we select the set of *Pareto* materializable global plan nodes for each schema. For each materialized view configuration generated by MOEA Framework, the simulator calculates performance and power consumption using cost models. Similarly, it then needs to select a unique view configuration with the desired trade-off: power-MV (*MV for materialized views*), time-MV, trade-off-MV, using weighted sum method with corresponding ω_i. Note that I/O costs were converted to time values (ms).

Our experiments (some of which are depicted in Fig16) show that (i) logical schemes intended to improve an NFR (performance/power), do not necessarily give the optimal values in the presence of optimization structures. That said, they do not give the worst values either, (ii) to orient a designer towards a given NFR at earlier stages, she/he

[17] Java library for multi-objective evolutionary algorithms. www.moeaframework.org.



(a) Oriented-tradeoff logical schema



(b) Oriented-time logical schema (c) Oriented-power logical schema

Fig. 16: Impact of $\mathcal{VM}$ on both Power and Performance of Physical Optimizations.

must combine the suitable tradeoff of both logical schema and optimization structures, (iii) these results confirm the need for a holistic variability-aware design process where such interdependences have to be considered.

6 Conclusion

In this paper, we discuss a challenging issue related to the integration of energy in the database world. We first summarize the initiatives that the database community did for building energy applications and DBMS. This discussion is based on four actions that we consider relevant which are: *Offer*, *Borrow*, *Reform* and *Evaluate*. According our analysis of the existing studies dealing with integrating the energy in the database technology and the well advanced domains in terms of energy management such as Green Building and Smart Cities, we propose two main initiatives to design existing and new databases/data warehouses. In the case, where a database is operational, we can easily interact with the query optimizer of the DBMS hosting it. It should be noticed that the query optimizers takes the lion's share of computation resources. As a consequence, we propose an energy-aware query optimizer deployed in *PostgreSQL DBMS*. The integration of the energy is performed by the means of the development of mathematical cost models dedicated to estimate the energy consumption when executing a set of queries. At this stage, we have an operational tool, called *EnerQuery*, available at the forge of our laboratory: `https://forge.lias-lab.fr/projects/ecoprod` and accepted in ACM-CIKM 2016 [54].

The second initiative is to encourage designers to consider the energy when constructing warehouse applications. Due to the complexity of all phases, we consider in this paper the logical phase because it is well connected to the physical one. As a consequence, we launched a think-tank about the impact of varying the logical model of a given $\mathcal{DW}$ on the physical design, according to two non-functional requirements: efficiency of energy consumption and query performance. This think-tank is alimented by

(a) a debate on the analogy between Software Product Lines (SPL) and $\mathcal{DB}$ design, **(b)** tools to identify/model the dimensions of our problem variability, **(c)** the efforts that designers have to make to deal with this interesting issue. To show the consequences of varying the logical schema on the physical design, we handled two scenarios: **(i)** a physical schema without physical optimization and **(ii)** a physical schema with the process of selecting materialized views. These two scenarios are evaluated using the Star Schema Benchmark and specific hardware to capture energy. The obtained results shows the worthiness of launching our think-tank that the frozen logical schema, and star schema in particular, is not always the best one to satisfy the fixed non-functional requirements.

Currently, we are working on pushing back the variability to cover other phases of the life cycle such as ETL and conceptual modeling.

References

1. D. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, et al. The beckman report on database research. *Communications of the ACM*, 59(2):92–99, 2016.
2. S. Abdellaoui, L. Bellatreche, and F. Nader. A quality-driven approach for building heterogeneous distributed databases: The case of data warehouses. In *IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 631–638, 2016.
3. A. Abelló, J. Samos, and F. Saltor. Yam2: a multidimensional conceptual model extending uml. *Information Systems Journal*, 31(6):541–567, 2006.
4. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.
5. Z. Akkaoui, J. Mazón, A. Vaisman, and A. Zimányi. Bpmn-based conceptual modeling of etl processes. In *DaWaK*, pages 1–14, 2012.
6. S. Anderlik, B. Neumayr, and M. Schrefl. Using domain ontolologies as semantic dimensions in data warehouses. In *ER*, pages 88–101, 2012.
7. S. Apel, D. Batory, C. Kstner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, 2013.
8. L. Bellatreche, S. Khouri, and N. Berkani. Semantic data warehouse design: From ETL to deployment à la carte. In *DASFAA*, pages 64–83, 2013.
9. S. Benkrid. *Le déploiement, une phase à part entière dans le cycle de vie des entrepôts de données : application aux plateformes parallèles*. PhD thesis, ISAE-ENSMA and ESI of Algeria, jun 2014.
10. N. Berkani, L. Bellatreche, and B. Benatallah. A value-added approach to design BI applications. In *DAWAK*, pages 361–375, 2016.
11. M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin, J. Gray, W. F. K. III, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, I. L. Traiger, B. W. Wade, and R. A. Yost. System R: an architectural overview. *IBM Systems Journal*, 38(2/3):375–396, 1999.
12. P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.
13. S. Bouarar, L. Bellatreche, S. Jean, and M. Baron. Do rule-based approaches still make sense in logical data warehouse design? In *ADBIS*, pages 83–96, 2014.
14. S. Bouarar, L. Bellatreche, and A. Roukh. Eco-data warehouse design through logical variability. In *43nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2017.

15. I. Boukhari. *Intégration et exploitation de besoins en entreprise étendue fondes sur la sémantique*. PhD thesis, ISAE-ENSMA, jan 2014.
16. A. Boukorca, L. Bellatreche, S. B. Senouci, and Z. Faget. Coupling materialized view selection to multi query optimization: Hyper graph approach. *IJDWM*, 11(2):62–84, 2015.
17. J. Bowen. *Getting Started with Talend Open Studio for Data Integration*. Packt Publishing Ltd, 2012.
18. P. G. Brown and P. J. Hass. Bhunt: automatic discovery of fuzzy algebraic constraints in relational data. In *VLDB*, pages 668–679, 2003.
19. A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. Securebpmn: modeling and enforcing access control requirements in business processes. In *ACM SACMAT*, pages 123–126, 2012.
20. D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In *Logics for Databases and Information Systems*, pages 229–263, 1998.
21. S. Chaudhuri, V. Narasayya, and R. Ramamurthy. Estimating progress of execution for sql queries. In *ACM SIGMOD*, pages 803–814. ACM, 2004.
22. L. Dannecker, R. Schulze, M. Böhm, W. Lehner, and G. Hackenbroich. Context-aware parameter estimation for forecast models in the energy domain. In *SSDBM*, pages 491–508. Springer, 2011.
23. G. e Sustainability Initiative and I. the Boston Consulting Group. Gesi smarter 2020: The role of ict in driving a sustainable future. Press Release, December 2012.
24. L. Etcheverry and A. A. Vaisman. Enhancing olap analysis with web cubes. In *The Semantic Web: Research and Applications*, pages 469–483. Springer, 2012.
25. C. H. Goh. Context interchange: New features and formalisms for the intelligent integration of information. *ACM TOIS*, pages 270–293, 1999.
26. M. Golfarelli, D. Maio, and S. Rizzi. Conceptual design of data warehouses from e/r schemes. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, pages 334–343. IEEE, 1998.
27. M. Golfarelli and S. Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, Inc., 1 edition, 2009.
28. P. Hitzler, M. Krotzsch, and S. Rudolph. *Foundations of semantic web technologies*. CRC Press, 2011.
29. J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. YAGO2: exploring and querying world knowledge in time, space, context, and many languages. In *WWW*, pages 229–232, 2011.
30. Intel and Oracle. Oracle exadata on intel® xeon® processors: Extreme performance for enterprise computing. White paper, 2011.
31. M. J-N. and J. Trujillo. An mda approach for the development of data warehouses. In *JISBD*, pages 208–208, 2009.
32. S. Khouri. *Cycle de vie sémantique de conception de systèmes de stockage et de manipulation de données*. PhD thesis, ISAE-ENSMA and ESI of Algeria, oct 2013.
33. H. Kimura, G. Huo, A. Rasin, S. Madden, and S. Zdonik. Coradd: Correlation aware database designer for materialized views and indexes. *PVLDB*, 3(1):1103–1113, 2010.
34. M. Kunjir, P. K. Birwa, and J. R. Haritsa. Peak power plays in database engines. In *EDBT*, pages 444–455. ACM, 2012.
35. W. Lang, R. Kandhan, and J. M. Patel. Rethinking query processing for energy efficiency: Slowing down to win the race. *IEEE Data Eng. Bull.*, 34(1):12–23, 2011.
36. W. Lang and J. Patel. Towards eco-friendly database management systems. *arXiv preprint arXiv:0909.1767*, 2009.

37. S. Lauesen. Task descriptions as functional requirements. *IEEE Software*, 20(2):58–65, Mar. 2003.
38. M. Levene and G. Loizou. Why is the snowflake schema a good data warehouse design? *Inf. Syst.*, 28(3):225–240, 2003.
39. S. Luján-Mora, J. Trujillo, and I.-Y. Song. A uml profile for multidimensional modeling in data warehouses. *Data & Knowledge Engineering Journal*, 59(3):725–769, 2006.
40. S. Luján-Mora, P. Vassiliadis, and J. Trujillo. Data mapping diagrams for data warehouse design with uml. In *ER*, pages 191–204, 2004.
41. I. Mami and Z. Bellahsene. A survey of view selection methods. *SIGMOD Record*, 41(1):20–29, 2012.
42. J. C. McCullough, Y. Agarwal, et al. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conf*, 2011.
43. E. Nakuçi, V. Theodorou, P. Jovanovic, and A. Abelló. Bijoux: Data generator for evaluating ETL process quality. In *ACM DOLAP*, pages 23–32, 2014.
44. V. Nebot and R. Berlanga. Building data warehouses with semantic web data. *Decision Support Systems*, 52(4):853–868, 2012.
45. E. Otoo, D. Rotem, and S.-C. Tsao. Energy smart management of scientific data. In *SSDBM*, pages 92–109. Springer, 2009.
46. T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. *Information Systems, Journal*, 26(5):383–423, 2001.
47. J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *ICDE*, pages 218–227, 1996.
48. M. Poess and R. O. Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. *PVLDB*, 1(2):1229–1240, 2008.
49. P. Ponniah. *Data Warehousing Fundamentals for IT Professionals*. Wiley, 2010.
50. W. Rasdorf, K. Ulberg, and J. Baugh Jr. A structure-based model of semantic integrity constraints for relational data bases. *In Proc. of Engineering with Computers*, 2:31–39, 1987.
51. M. Rodriguez-Martinez, H. Valdivia, et al. Estimating power/energy consumption in database servers. *Procedia Computer Science*, 6:112–117, 2011.
52. A. Roukh and L. Bellatreche. Eco-processing of olap complex queries. In *DaWaK*, pages 229–242, 2015.
53. A. Roukh, L. Bellatreche, A. Boukorca, and S. Bouarar. Eco-dmw: Eco-design methodology for data warehouses. In *DOLAP*, pages 1–10. ACM, 2015.
54. A. Roukh, L. Bellatreche, and C. Ordonez. Enerquery: Energy-aware query processing. *To appear in ACM CIKM*, 2016.
55. K. Royer, L. Bellatreche, et al. One semantic data warehouse fits both electrical vehicle data and their business processes. In *ITSC*, pages 635 – 640, 2014.
56. C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the e/r model for the multidimensional paradigm. In *Advances in Database Technologies*, pages 105–116. Springer, 1999.
57. O. Shmueli and S. Tsur. Logical diagnosis of ldl programs. *New Generation Computing*, 9(3/4):277–304, 1991.
58. L. Siksnys, C. Thomsen, and T. B. Pedersen. MIRABEL DW: managing complex energy data in a smart grid. In *DAWAK*, pages 443–457, 2012.

59. A. Simitsis. Mapping conceptual to logical models for etl processes. In *Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, pages 67–76. ACM, 2005.
60. A. Simitsis, P. Vassiliadis, and T.-K. Sellis. Optimizing etl processes in data warehouses. In *ICDE*, pages 564–575, 2005.
61. A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos. Optimizing etl workflows for fault-tolerance. In *ICDE*, pages 385–396, 2010.
62. D. Skoutas and A. Simitsis. Ontology-based conceptual design of ETL processes for both structured and semi-structured data. *Int. J. Semantic Web Inf. Syst.*, 3(4):1–24, 2007.
63. T. Stöhr, H. Märtens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. In *VLDB*, pages 273–284, 2000.
64. A. Tort, A. Olivé, and M. Sancho. An approach to test-driven development of conceptual schemas. *Data Knowl. Eng.*, 70(12):1088–1111, 2011.
65. J. Trujillo and S. Luján-Mora. A uml based approach for modeling etl processes in data warehouses. In *ER*, pages 307–320, 2003.
66. J. Trujillo, M. Palomar, J. Gomez, and I.-Y. Song. Designing data warehouses with oo conceptual models. *IEEE Computer*, 34(12):66–75, 2001.
67. P. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the physical implementation of etl workflows. In *ACM DOLAP*, pages 49–56, 2007.
68. A. Vaisman and E. Zimányi. *Data Warehouse Systems: Design and Implementation*. Springer, 2014.
69. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of etl scenarios. *Inf. Syst.*, 30(7):492–525, 2005.
70. P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for etl processes. In *DOLAP*, pages 14–21, 2002.
71. P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for etl processes. In *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, pages 14–21. ACM, 2002.
72. P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Modeling etl activities as graphs. In *DMDW*, pages 52–61, 2002.
73. K. Wilkinson, A. Simitsis, M. Castellanos, and U. Dayal. Leveraging business process models for etl design. In *ER*, pages 15–30, 2010.
74. Z. Xu, Y.-C. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *ICDE*, pages 485–496, 2010.
75. Z. Xu, Y.-C. Tu, and X. Wang. Dynamic energy estimation of query plans in database systems. In *ICDCS*, pages 83–92. IEEE, 2013.
76. A. Zhou, B. Qu, H. Li, S. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation, Elsevier*, 1(1):32–49, 2011.