EnerQuery: Energy-Aware Query Processing

Amine Roukh
University of Mostaganem
Mostaganem, Algeria
roukh.amine@univ-mosta.dz

Ladjel Bellatreche
LIAS/ISAE-ENSMA
Poitiers, France
bellatreche@ensma.fr

Carlos Ordonez
University of Houston
Houston, USA
ordonez@cs.uh.edu

ABSTRACT

Energy consumption is increasingly more important in large-scale query processing. This requires to revisit the traditional query processing in actual DBMSs to identify the potential of energy saving, and to study the trade-offs between energy and performance. In this paper, we propose a tool, called *EnerQuery* built on PostgreSQL DBMS to capitalize on the efforts put into building energy-aware query optimizers, which have the lion's share of the overall energy consumption. The energy is projected on all PostgreSQL query optimizer steps and integrated into its mathematical cost models used to select the best query plans. To increase end users' energy awareness, we connect a diagnostic tool with user-friendly interface to *EnerQuery* to visualize the energy consumption and its savings when tuning some parameters during the query execution process.

1. INTRODUCTION

Nowadays, with the deluge of data, emerging hardware technologies and the query efficiency requirements of developers and users of database applications, data centers are in the line of sight, because they increase the greenhouse gas emissions. In a typical data center, DBMS is one of the most avid consumers in terms of computational resources among other software deployed [5]. As researchers working in the database field, we must propose an initiative that integrates energy into DBMS functionalities and gives developers and end users the possibility to see the energy consumption when executing their queries. In this demonstration, we only focus on the query optimizer which is one of the main DBMS components. A past work [4] argues that optimizing for performance always leads to a high energy efficiency. However, in their study the authors considered the idle power of the system under test, which limits the potential for active power savings. Moreover, recent studies have shown that there exist opportunities for an energy efficient query optimization in current DBMSs [2, 5]; this is also proven in our finding.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

XXXX 'XX Date xx–xx, 20xx, Xxxxx, XX, XXX

© 2016 ACM. ISBN xxx-xxxx-xx-xxx/xx/xx…$15.00

DOI: 10.475/123_4

In this paper, we propose an energy-aware query optimizer framework, called EnerQuery[1], which leverages the PostgreSQL query optimizer by integrating the energy dimension. To ensure this integration, we fix three main objectives: **(i)** the identification energy-sensitive layers of the query optimizer, **(ii)** the definition of mathematical cost models estimating the energy consumption of SQL queries, and **(iii)** the development of a graphical user interface that plays the role of a diagnostic tool for end users, developers and DBA to increase their energy awareness and pushing them to save it. These three objectives will be discussed in the next parts of the paper.

2. ENERGY-AWARE QUERY PROCESSING

The process of executing a given query passes through four main steps: **(i)** parsing, **(ii)** rewriting, **(iii)** planning/optimizing, and **(iv)** executing. The *parser* checks the query string for valid syntax using a set of grammar rules, then translates the query into an equivalent relational algebra expression. The output is the *query tree*. The *rewriter* processes and rewrites the tree using a set of rules. In our demonstration, the energy cost of query parser and rewrite is *negligible*. The *planner/optimizer* creates an optimal execution plan. A given SQL query can be executed in different ways while producing the same results. The optimizer's task is to estimate the cost of executing each plan using a mathematical cost model and to find out the fast one. In *EnerQuery*, the optimizer evaluates a query plan by considering two metrics: the query processing time and the energy consumption.

The cost model is composed of functions for each basic SQL operator. Generally, the cost formula used to estimate the cost of an operation *op* is: $Cost_{op} = \alpha \times I/O \oplus \beta \times CPU \oplus \gamma \times Net$. Where I/O, CPU, and Net are respectively the estimated page numbers, tuple numbers, and communication messages required to execute *op*. They are usually computed using database statistics and selectivity formulas. The coefficients α, β and γ are used to convert estimations into the desired unit (e.g., time, energy). $\oplus$ represents the relationship between the parameters (linear, non-linear). The coefficient parameters and the relationship can be obtained using various techniques such as calibration, regression, and statistics.

The *executor* takes the plan created by the planner/optimizer and processes it recursively to extract the required set of

[1]A video demonstration of *EnerQuery* features is available at: https://youtu.be/cWK5rf4MBNQ
A download link of the tool can be found at: http://www.lias-lab.fr/forge/projects/ecoprod

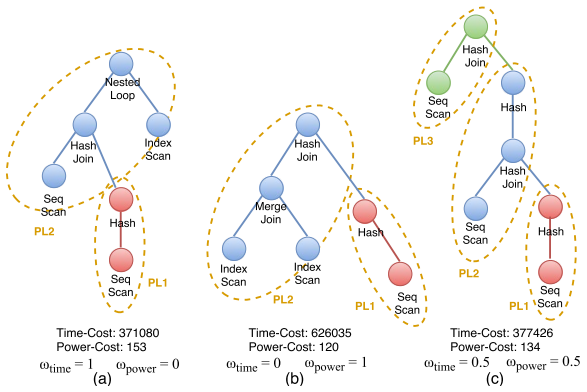

Figure 1: Segmented plans of query Q3 associated to trade-off values.

rows. This is essentially a demand-pull *pipeline* (PL) mechanism (called *iterator* or *GetNext* model in [1]).

To illustrate this mechanism supported by our *EnerQuery*, we consider the execution of the query *Q3* of the TPC-H benchmark[2]. Figure 1 gives three possible execution plans of *Q3* returned by *EnerQuery* based on the trade-off values assigned to the query response time and the energy consumption. Note that the time and power costs are different and impacted by the *model of pipelined execution* used by the DBMS [3]. Usually, this model segments each query plan into a set of PLs. In Figure 1, the plans **(a)** and **(b)** both have two PLs, whereas the plan **(c)** has three PLs. The execution of PLs is enforced by their terminal *blocking* operators (e.g., PL3 cannot begin until PL2 is complete).

EnerQuery is guided by a PL-based power cost model which delivers a high estimation accuracy. The end user *is free to guide* the process of evaluating execution plans of a given query by setting up the trade-off between the query response time and the energy consumption. *EnerQuery* is associated with a graphical user interface that offers the possibility to interact with the DBMS, to calibrate parameters of the cost model, and to identify the energy consumption of each part of the query being executed.

3. SYSTEM ARCHITECTURE

In this section, we describe the architecture of *EnerQuery*. It is composed of two parts, the backend and the graphical user interface. The backend contains the DBMS with the new power cost model and the evaluation model. It is responsible for executing queries with the desired trade-off and returning results to end users. In our study, we based on PostgreSQL, mostly because it is open source. However, our techniques can be integrated into any other DBMS. The GUI part helps to manipulate *EnerQuery*, to set parameters and to show in real-time their impact on the power consumption. The framework architecture of our demonstration is described in Figure 2. In the following sections, we will briefly present our underlying models. Full details can be found in [3].

3.1 Power Consumption Linear Cost Model

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

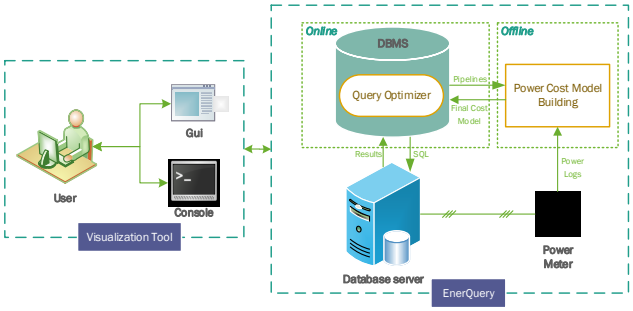

Figure 2: *EnerQuery* system architecture.

The characteristics of our model dedicated to estimating queries power include: **(i)** the segmentation of an execution plan into a set of PLs, **(ii)** the utilization of the PL cost to build the regression model (*off-line*), and **(iii)** the estimation of the power of future PL based on PL cost and the regression equation (*on-line*).

3.1.1 Pipeline Segmentation

A physical operator of a query execution plan can be either *blocking* or *nonblocking*. An operator is blocking if it cannot produce any output tuple without reading as least one of its inputs (e.g., sort operator) [1]. Based on this, we decompose a plan in a set of PLs delimited by blocking operators. Thus, a PL consists of a set of concurrently running operators, as showed in Figure 1.

3.1.2 Model Parameters

Our strategy for PL modeling is to leverage the PostgreSQL cost models by integrating energy. To process a query, each operator in a PL needs to perform CPU and/or I/O tasks. The cost of these tasks is the *active power* to be consumed in order to finish them. More formally, the power cost $Power(Q)$ of the query Q composed of p pipelines is: $Power(Q) = \frac{\sum_{i=1}^{p} Power(PL_i) * Time(PL_i)}{Time(Q)}$. The *time* variable represents the PLs and the query estimated time to finish the execution. These informations are available directly from the DBMS. The power cost $Power(PL_i)$ of the pipeline PL_i composed of n algebraic operations is the summation of CPU and I/O costs of all its operators: $Power(PL_i) = \beta_{cpu} \times \sum_{j=1}^{n_i} CPU\_COST_j + \beta_{io} \times \sum_{j=1}^{n_i} IO\_COST_j$, where β_{cpu} and β_{io} are the model parameters (i.e., unit power costs) for the PLs. These parameters are computed using multiple polynomial regression techniques, which describes perfectly the behavior of PLs power consumption on the hardware components (disk and CPU) of our platform [3]. The β parameters are estimated during the learning phase from a training data, by finding the least-squares solution.

3.2 Plans Evaluation

Adding energy criterion to the query optimizer, we must adjust the comparison functions to reflect the trade-offs between energy cost and processing time. In order to give a solution with the desired trade-off, we use the weighted sum of the cost functions method to aggregate criteria and to have an equivalent single criterion to be minimized. This method is defined as follows: minimize $y = f(x) = \sum_{i=1}^{k} \omega_i \cdot f_i(\vec{x})$ such that $\sum_{i=1}^{k} \omega_i = 1$, where ω_i are the weighting coefficients representing the relative importance of the k

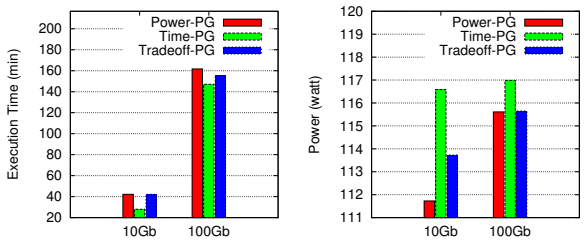

Figure 3: Performance and power saving with different *EnerQuery* configurations using TPC-H.

cost functions. $f_i(x)$ represents power and performance cost functions. We implemented these two coefficients as an external parameter in *EnerQuery*, so the DBA or users can change them on-the-fly either by the GUI or the console.

4. DEMONSTRATION SCENARIO

In our demonstration, we will use a "Watts UP? Pro ES" power meter to measure the laptop power, which acts like a server and sets up the *EnerQuery* framework. We will use the TPC-H benchmark datasets and queries, and the SSB benchmark by varying their size. During the demonstration, we will highlight the role of *EnerQuery* in evaluating the effectiveness and efficiency of our proposal in several prominent scenarios. In practice, we study the effect of **(i)** varying the optimization goal of the query planner, **(ii)** varying dataset and queries, and **(iii)** showing the estimation and actual power consumption. Moreover, in each scenario, we will visualize the actual power consumption of the system and show the total energy consumption, as well as the query performance. Based on these scenarios, we can study and compare query operators behavior on power and performance results.

We will also use the *EnerQuery* GUI interfaces to allow the user manipulating the framework settings and seeing their impact on the system. The interface is implemented using C++ programming language and Qt library, which provides a better integration with the DBMS. Figure 1 illustrates a scenario where different query plans of the same query generated by *EnerQuery*: **(a)** is performance-oriented, **(b)** is power-oriented, and **(c)** is a trade-off. Users can analyze the costs of each operator using *EnerQuery* GUI, they can realize that the joins have a major impact on the nature of query plan. Figure 3 represents a scenario of the previous experiment extended to the overall TPC-H query workload (details can be found in [3]). The main GUI component modules are listed below.

4.1 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 parametrize the optimization goals: performance or power or trade-off. Users can also *change* the query planner configuration parameters by *forcing* the optimizer to evaluate other plans (see screenshot ①), as Oracle *hints* do. These parameters cover the following optimization modes: *sequential*, *index*, *index-only*, *bitmap*, and *TIDs* scan types, *hash*, *merge*, and *nested loop* join types, *sort* and *hash aggregate*. *EnerQuery* offers end users the possibility to quantify the consumed power of a given query by varying the optimization modes.

4.2 SQL Query Workload

In this module, users can give either a single SQL query or workload to be executed. Queries supported vary from simple transactional operations to more complex reporting operations involving several large size tables. The workload is composed of queries generated from the benchmarks tools and can be run concurrently at a predefined multi-programming level. The execution is done in a separate thread for each query and the results are displayed in a tree table widget. An example of query that users can introduce is *Q7* of TPC-H, which is a nested query of two levels that joins 7 tables; it also contains a complex ordering and grouping operations.

4.3 Power Timeline

When the user executes a query, we will dynamically display the real-time power consumption via the power meter. When the query finishes executing, we will compute and show the total energy that has been consumed. This can give users a real observation of the energy that has been saved using the desired trade-off parameters. For example, in Figure 1 (c), for a 1.7% performance degradation, we get 12.4% of power saving. Also, users can compare the estimated and the real values to check the accuracy of mathematical models or further refine their parameters.

4.4 Execution Plan

When the user submits a query, the query optimizer selects its best execution plan relative to the pre-defined trade-off. We will show this execution plan with various information, such as estimated cost, power consumption, I/O and CPU costs for every physical operator through mouse-hovering events. Users can identify which operator consumes more power, for instance, we will show that CPU intensive operators such as sort and aggregate are power hungry in traditional servers. Moreover, we will highlight in which case I/O intensive operators lead to high power consumption. Also, the PL notation will be demonstrated as shown in Figure 1, the PL trees are grouped with the *same color*. We will show how the trade-off parameters affect the generated plans. This helps interpret runtime optimizations and visualize different PLs.

5. REFERENCES

[1] S. Chaudhuri, V. Narasayya, and R. Ramamurthy. Estimating progress of execution for sql queries. In *ACM SIGMOD*, pages 803–814. ACM, 2004.

[2] 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.

[3] A. Roukh, L. Bellatreche, A. Boukorca, and S. Bouarar. Eco-dmw: Eco-design methodology for data warehouses. In *DOLAP*, pages 1–10. ACM, 2015.

[4] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *sigmod*, pages 231–242, 2010.

[5] Z. Xu, Y.-C. Tu, and X. Wang. Pet: reducing database energy cost via query optimization. *Proceedings of the VLDB Endowment*, 5(12):1954–1957, 2012.

APPENDIX

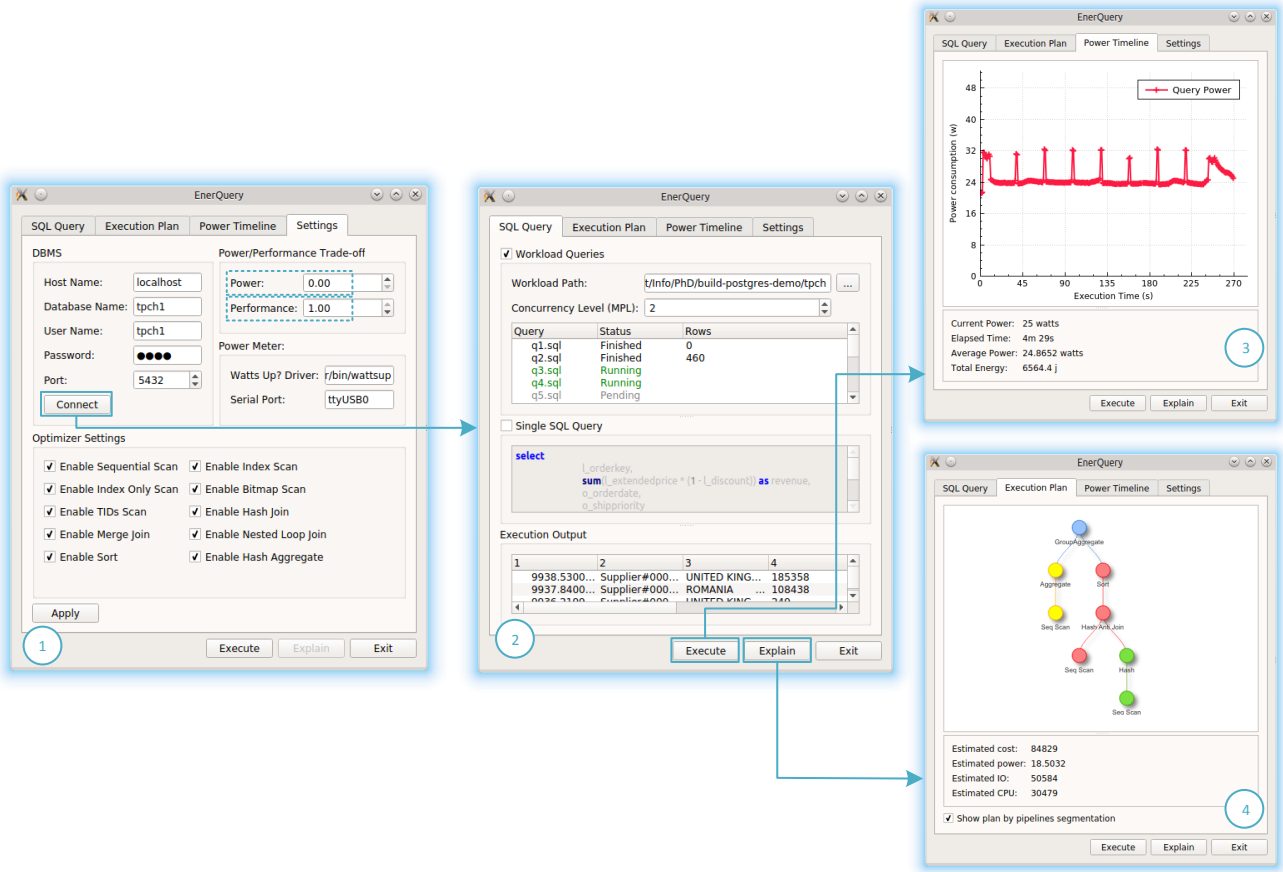

Figure 4: EnerQuery main GUI and its component module panels.