Energy-Aware Query Processing on Parallel Database Cluster Nodes

Amine Roukh[1], Ladjel Bellatreche[2], Nikos Tziritas[3], and Carlos Ordonez[4]

[1] University of Mostaganem, Mostaganem, Algeria
roukh.amine@univ-mosta.dz
[2] LIAS/ISAE-ENSMA, Poitiers University, France
bellatreche@ensma.fr
[3] Chinese Academy of Sciences, Shenzhen, China
nikolaos@siat.ac.cn
[4] Houston University, USA.
ordonez@cs.uh.edu

Abstract. In the last few years, we have been seeing a significant increase in research about the energy efficiency of hardware and software components in both centralized and parallel platforms. In data centers, DBMSs are one of the major energy consumers, in which, a large amount of data is queried by complex queries running daily. Having green nodes is a pre-condition to design an energy-aware parallel database cluster. Generally, the most existing DBMSs focus to high-performance during query optimization phase, while usually ignoring the energy consumption of the queries. In this paper, we propose a methodology, supported by a tool called *EnerQuery*, that makes nodes of parallel database clusters saving energy when optimizing queries. To show its effectiveness, we implement it in query optimize of PostgreSQL DBMS. A mathematical cost model is used to estimate the energy consumption. Its parameters are identified by a machine learning technique. We conduct intensive experiments using our cost models and a measurement tool dedicated to compute energy using dataset of TPC-H benchmark. Based on the obtained results, a probabilistic proof to demonstrate the confidence bounds of our model and results is given.

1 Introduction

The COP21[5] event shows the willingness of countries (Over 145 foreign Heads of State and Government attended the conference at Le Bourget, Paris), companies, individuals, government and non-governmental associations, etc. to save the planet. The continued expansion of the industry means that the energy used by data centers, and the associated emissions of greenhouse gases 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 [1]. In a typical data center, DBMS is one of the most important consumers of computational resources among other software deployed, which turn DBMS to be a considerable energy consumer [2]. Traditionally, the design

[5] http://www.gouvernement.fr/en/cop21

process of a database considers one non-functional requirement, which represents the query response time. This requirement is quite comprehensive, since the end user and decision makers of database applications are looking for the efficiency of the queries. Note that in the *Beckman report* on databases published in last February, energy constrained processing and scientific data management are considered as challenging issues [3].

Face to the strong requirement of saving energy, database community did not stand idly, but from last decade, it continuously proposes initiatives covering centralized and parallel and distributed databases [4, 5]. These initiatives have shown their performance in reducing energy consumption. In this paper, we concentrate on a node of a parallel database cluster storing and managing data warehouses [6]. To ensure the performance of a node in terms of energy consumption, the different components of its DBMS have to integrate energy when executing queries. Note that the landscape of DBMS is very large; since it includes several components: query optimizer, storage manager, etc. In this paper, we focus on query optimizers, which represents one of the main components of DBMS. There has been extensive work in query optimization since the early'70 in traditional databases. Several algorithms and systems have been proposed, such as *System-R project*, where its findings have been largely incorporated in many commercial optimizers. Advanced query optimizers perform two main tasks: **(i)** enumeration of execution plans for a given query and **(ii)** the selection of the best plan. The existing studies on energy-aware query optimizers consider mainly the second task, by reforming the cost models to integrate energy.

In this paper, we focus on the query optimization component of the PostgreSQL DBMS. We propose a design methodology, supported by a tool called *EnerQuery*[6], that tries to integrate energy in the query generation phase. This is done by revisiting all the query optimizer steps and studying their effect on energy consumption. The new query optimizer will have to deal with two objective functions, namely: improving performance and minimizing energy. In our design, the end users can specify preferences in their profiles by setting weights on different objectives, representing relative importance. The role of the *EnerQuery* is to minimize the weighted sum over different cost metrics.

The main technical contributions of this paper are: **(i)** a multi-objective formalization of the query optimization problem including the query performance and the energy consumption within a cluster node; **(ii)** intensive experiments using real tools to study the effectiveness of our approaches and **(iii)** a probabilistic proof is given to demonstrate the confidence bounds of our model data and results, using high-end configuration experimentation data.

Our paper is organized as follows. Section 2 presents the related work. Section 3 describes our green query optimizers. Section 4 shows and interprets our experimental results. Section 5 gives a probabilistic complexity study to demonstrate the confidence bounds of our finding while our conclusion is given in Section 6.

2 Related Work

Recently, there has been a plethora of work by the research community in the field of energy-efficiency optimizations in database systems covering hardware [7] and software levels. This section reviews only the research efforts related on software aspects that mainly concern the definition of cost models estimating the energy consumption and their usage in proposing optimization techniques.

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

In [8,9], 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. In [10], a technique for modeling the peak power of database operations is given. 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 [11] 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. [12] attempts to model energy and peak power of simple selection queries on single relations using linear regression. In our previous works [13], 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. The presence of energy consumption cost models motivates the research community to propose cost-driven techniques. The work in [14] 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 [11] 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 [10], a cost-based driven approach is proposed to generate query plans minimizing the peak power. In [15], a genetic algorithm with a fitness function based on an energy consumption cost model, is given to select materialized views reducing energy and optimizing queries. The work by Xu *et al.* [9] 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 does not capture the relationship between the parameters.

3 Energy-Aware Query Processing

In order to build energy-aware query optimizers, we first propose an audit of each component to understand whether it is energy-sensitive or not. After our audit, we present in details our methodology to construct our optimizer.

3.1 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. To illustrate these steps, we consider PostgreSQL DBMS as a case study.

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 a 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*.

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.

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[7]. This a complex query which involves the join of 7 tables. 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.

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

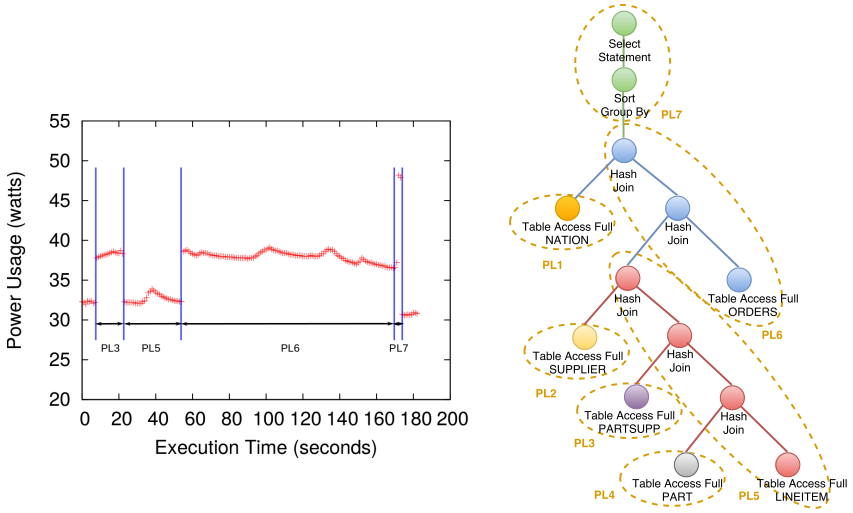
Considering this small number of tables, if we go in real operational databases where there are a hundred of tables, the searching strategy used by the planner can lead to a noticeable energy consumption. Setting the query plan of queries manually by the database administrator is recommended in large 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:

$$Cost_{op} = \alpha \times I/O \oplus \beta \times CPU \oplus \gamma \times Net \quad (1)$$

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.

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.



(a) Pipelines segmented power consumption (b) Pipelines segmented execution plan

Fig. 1. Power consumption and execution plan of TPC-H benchmark query *Q9* with corresponding pipeline annotation.

Example 1. To study the effect of the execution step on designing green-query optimizer, we consider an example of query *Q9* from the TPC-H benchmark. Figure 1b presents the execution plan returned by the query optimizer and Figure 1a shows the active power consumption during the execution of query [13]. As we can see from the figure, the power consumption is directly influenced by execution model of the DBMS. 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 query *Q9* execution plan is shown in Figure 1b, there are 7 pipelines (only 4 are important in our discussion because the others end faster), and a partial order of the execution of these pipelines is enforced by their terminal *blocking* operators (e.g, PL6 cannot begin until PL5 is complete). We can also see from Figure 1 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 [13]. Therefore, the pipelining execution is very important and has a direct impact on power consumption during query execution. The design of power cost model should take into consideration the execution strategy, which is ignored by Xu *et al.* [9].

3.2 Our Methodology

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 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 2.

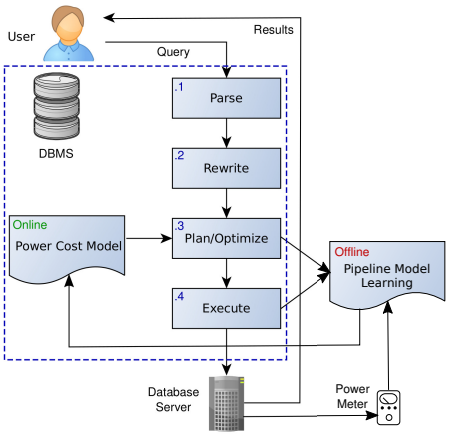


Fig. 2. The Design Methodology

3.3 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 parameters to build the model by employing machine learning techniques (*off-line*), and (iii) the estimation of the power of future pipeline based on pipeline parameters and the final model (*on-line*).

Pipeline Segmentation. When a query is submitted to the DBMS, the query optimizer chooses an execution plan (cf. Figure 1b). 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 [16]. As in previous work [16], 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 1b.

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 database systems 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:

$$Power(Q) = \frac{\sum_{i=1}^{p} Power(PL_i) * Time(PL_i)}{Time(Q)} \tag{2}$$

The *time* function represents the pipelines and the query estimated time to finish the execution. Unlike Xu *et al.* study which ignores the execution time [8], 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:

$$Power(PL_i) = \beta_{cpu} \times \sum_{j=1}^{n_i} CPU\_COST_j + \beta_{io} \times \sum_{j=1}^{n_i} IO\_COST_j \tag{3}$$

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. A summary of the formulas used to calculate I/O and CPU costs for each basic operator can be found in Table 3b with the symbols listed in Table 3a.

Table 2. Cost model calculation formulas and parameters.

Parameter	Definition
β_{cpu}	power to perform one I/O op
β_{io}	power to perform one CPU op
m	buffer memory size
$block$	DBMS page size
T_i	the size of table i
t_i	# of input tuple for the op i
f	index selectivity
s	input relation size for the sort op
n_{hash}	# of clauses in building phase
p_{hash}	# of partitions in probing phase
$p_{outer/inner}$	# of pages retrieved for join op
$t_{outer/inner}$	# of tuples retrieved for join op
n_{group}	# of grouping columns

(a) Cost model parameters notation.

Parameter	I/O Cost	CPU Cost
Sequential scan	p_{seq}	t_{seq}
Index scan	p_{index}	$t_{index} \cdot f$
Bitmap scan	p_{bitmap}	$t_{bitmap} \cdot f$
Nested loop join	$p_{outer} + p_{inner}$	$t_{outer} \cdot t_{inner}$
Sort merge join	$p_{outer} + p_{inner}$	$t_{sort(outer)} + t_{sort(inner)}$
Hash join	p_{outer}	$t_{outer} \cdot n_{hash} + t_{inner} \cdot p_{hash}$
Sort	$p_{sort}, s < m;$ $0, else$	$t \cdot log_2(t)$
Aggregate	0	$t_{agg} \cdot$
Group by	0	$t_{group} \cdot n_{group}$

(b) Cost model parameters for SQL operators.

Parameters Calibration. The key challenge in equation (3) is to find model parameters β_{cpu} and β_{io}. Simple linear regression technique, as used in [8, 10, 11], 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 (more details and results can be found in [13]) 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). To compute the power of pipelines we use the same formulas elaborated in our earlier work [13].

The β parameters of Equation 3 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 [17].

3.4 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 trade-offs 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 criteria and have an equivalent single criterion to be minimized. This method is defined as follows:

$$\text{minimize } y = f(x) = \sum_{i=1}^{k} \omega_i \cdot f_i(\overrightarrow{x}) \quad \text{such that } \sum_{i=1}^{k} \omega_i = 1 \tag{4}$$

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.

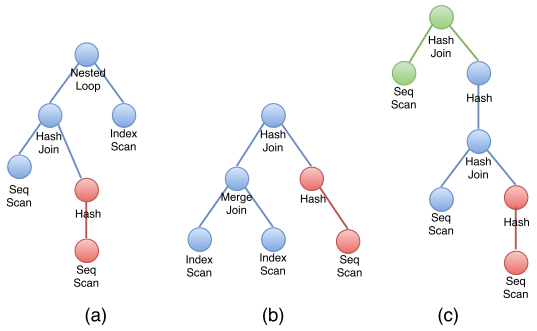


Fig. 3. The optimal plan for TPC-H query *Q3* when changing user preferences.

Figure 3 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 3a, 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 3b, 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 3c give a good trade-off, for a 1.7% of performance degradation, we get 12.4% of power saving.

4 Experiments and Results

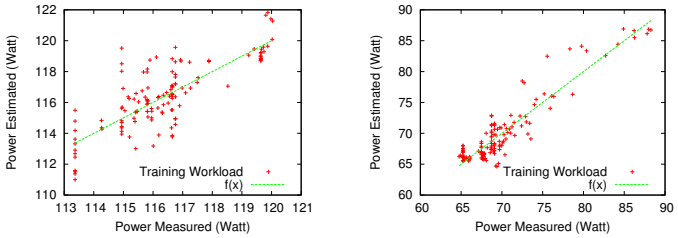
To evaluate the effectiveness of our proposal, we conduct several experiments. Next, we present our experimental machine to compute the energy and the used datasets and simulator.

Experiment Setup We use a similar setup used in the state-of-the-arts [8, 10, 11]. Our machine is equipped with a "Watts UP? Pro ES"[8] power meter with one second as a maximum resolution. The device is directly placed between the power supply and the database workstation under test to measure the workstation's overall power consumption. The power values are logged and processed in a separate monitor machine. We used a Dell PowerEdge R310 workstation having a Xeon X3430 2.40GHz processor and 32GB of DDR3 memory. To validate our model using different low-end hardware configuration, we created another setup with a Dell Precision T1500 workstation equipped with an Intel Core i5 2.27GHz processor and 4GB of memory. Our workstation machine is installed with our modified version of PostgreSQL 9.4.5 DBMS under Ubuntu 14.04 LTS with kernel 3.13. We use TPC-H datasets and queries with 10GB and 100GB scale factor. The TPC-H benchmark illustrates decision support systems that examine large volumes of data, execute different types of queries with a high degree of complexity. The queries are executed in an isolated way. In our experiments, we consider three types of PostgreSQL configuration: (1) Power-PG, which is the configuration that gives the minimal power cost, (2) Time-PG, is the configuration with minimal time cost, (3) Tradeoff-PG, using weighted sum method with $\omega_1 = 0.5$, $\omega_2 = 0.5$.

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

Power Model Building As mentioned above, the β parameters are estimated while learning the model from training data. We then perform a series of observations in which queries are well-chosen, and the power values consumed by the system are collected using a measurement equipment while running these queries. In the same time, for each training instance, we calculate their costs. To generate training instances, we create our custom query workload based on TPC-H datasets. The workload queries are divided into two main categories: **(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 a single table scan, queries with multiple joins with different predicates. They also contain sorting/grouping conditions and simple and advanced aggregation functions as in [10]. After collecting power consumption training queries, we apply the regression equation using the *R language software*[9] to find our model parameters. Once we get them, an estimation of new queries is obtained without the use of our measurement equipment.

Results In this paragraph, the set of results are commented.



(a) Regression Model fit using high-end configuration

(b) Regression Model fit using low-end configuration

Fig. 4. Training workload power consumption and regressions fit.

Cost Model Building Quality. To check the portability of our proposed cost model against changes in hardware environment, we conduct this experiment. The results of the training phase in our two setup configurations, against the fitted values from polynomial models are plotted in Figure 4. As we can see, the predicted and actual power consumption approximate the diagonal lines closely using our cost model in both configurations. Otherwise, in the server configuration, we can see some variance between the predicted and the observed power for some training queries. Much of this can be attributed to the errors made by the DBMS query optimizer in estimating IO and CPU costs for these queries when there is a large working memory. This problem has been faced by query optimizers for a long time, and all the performance models proposed so far suffer from this problem, which is inherited from the cardinality estimation errors. In fact, the estimation errors in the low level pipelines are propagated to the upper level and may significantly degrade the prediction accuracy.
Cost Model Estimation Error. In this type of experiment, given the estimated power cost predicted by our model *(E)*, we compare it with the actually observed system active power consumption *(M)*. To quantify the model accuracy, we used the following error ratio metric: $Error = \frac{|M-E|}{M}$. To test our model with large datasets, we run all 22 queries of the TPC-H benchmark against two database scale factor: 10GB and 100GB.

[9] http://www.r-project.org/

Most of the queries contain more than 4 pipelines. The results are shown in Table 3. Note that some queries were aborted since they exceeded 72 hours of execution in our current test environment.

Table 3. Estimation errors in TPC-H benchmark queries with different database sizes.

Query	10GB	100GB	Query	10GB	100GB
$Q1$	0.01	0.002	$Q11$	0.04	-
$Q2$	-	-	$Q12$	0.009	0.00029
$Q3$	0.01	0.01	$Q13$	0.04	0.04
$Q4$	0.006	0.005	$Q14$	0.02	0.02
$Q5$	0.01	0.03	$Q15$	0.004	0.02
$Q6$	0.04	0.02	$Q16$	0.05	0.0003
$Q7$	0.004	0.01	$Q18$	0.004	-
$Q8$	0.0007	0.01	$Q19$	0.01	0.009
$Q10$	0.006	0.003	$Q22$	0.01	0.004

As we can see from the table, the average error is typically small (0.1% in both 100GB and 10GB datasets), and the maximum error is usually below 5%. The experiment shows the accuracy of our prediction model, indicating that is sufficiently accurate for the intended applications.

Query Characterization. To study the characterization of the TPC-H 22 query, we conduct a series of tests using the modified PostgreSQL. In such tests and for each configuration (Time-PG, Power-PG, Tradeoff-PG) we run all the TPC-H queries and collect the estimated performance cost and power cost returned by the query optimizer. From Figure 5 (values are plotted on a logarithmic scale) we can see that 16 of 22 queries have the potential for power saving in the Power-PG configuration. Normally, the benefit of power saving for these queries has a negative impact on the processing time cost as shown in the same figure. However, choosing the trade-off configuration can lead to good power saving values with less performance degradation. These queries are characterized by an important number of SQL operators and various I/O and CPU operations, which gives the query optimizer a variety of plans to choose from. Therefore, we can achieve good power saving queries from those plans. On the other hand, the rest of queries that do not show opportunities for power saving, are simple queries with a few tables and SQL operators. This leads the query optimizer to choose the same plan in every PostgreSQL configuration, due to the small search space of the plans.

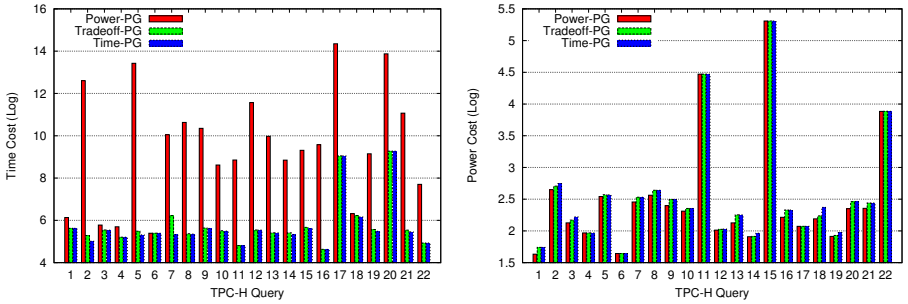


Fig. 5. Performance and power for TPC-H queries using different PostgreSQL configurations.

Power Saving. The purpose of this set of experiments is to investigate the benefit of

our approach in terms of energy efficiency. We configured the DBMS to evaluate the performance and power consumption cost models for the three configurations (Time-PG, Power-PG, Tradeoff-PG). We repeat the same experiments under two different database sizes: 10GB, and 100GB using TPC-H benchmark. In Figure 6 we present the results of the experiments. We can clearly see that workloads consume significantly lower power when choosing a query optimizer configuration that favors low-power plans. When comparing the power-only (Power-PG) with the performance-only (Time-PG) results, we observe a large margin in power savings, the benefit is remarkably considerable in small database size, perhaps this is due to the large amount of I/O operations and data processing required by queries of big database size which translate in more power consumption regardless of the chosen plan by query optimizer. As expected, the savings of the Tradeoff-PG configuration are smaller than those obtained by the power-only experiment, but it is still acceptable, especially, in 100GB datasets they are approximate. On the other hand, the power-only configuration takes more time to finish executing all the queries, which translate in a noticeable performance degradation. The above is not surprising, since if we gain in power we automatically lose in performance. In the Tradeoff-PG configuration, the performance degradation is actually acceptable if we consider the power gain achieved. Note that all results of our experiments only considered direct power savings in a single database server. This number could be even higher if we consider large-scale data centers with thousands of servers and cooling systems.

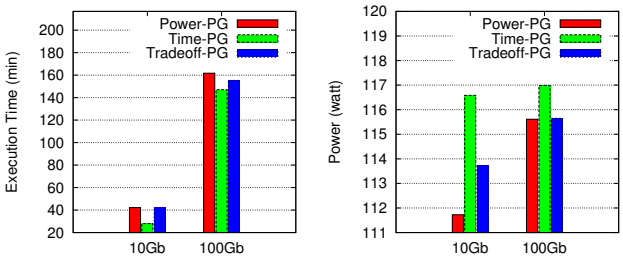


Fig. 6. Performance and power saving with different PostgreSQL configurations using TPC-H benchmark.

5 Confidence Bounds

In this section we prove the confidence bounds based on high-end configuration experimentation data. Finding confidence bounds is important in estimating the precision of an estimate [18]. So, we will prove the confidence in our model data and results, using high-end configuration experimentation data. To find the lower and upper bounds of population, we use Chebyshev's inequality (6). The inequality is based on population mean (denoted by μ) and population standard deviation (denoted by σ). Unfortunately, μ and σ are unknown parameters. Therefore, we must found their upper and lower bounds with some degree of confidence to calculate Chebyshev's inequality. To do the above, at Step 1, we test whether the samples come from a population that follows the normal distribution; at Step 2, we find the lower and upper bounds of the population mean, with the degree of confidence being 99%; at Step 3, we find the lower and upper bounds of the population standard deviation, with the degree of confidence being 99%; and at Step 4, we find the desired bounds.

Step 1. The sample mean equals 116.4554, which is denoted by $\bar{x}$; while the sample standard deviation equals 2.1822, which is denoted by s. The number of samples equals 131 and is denoted by n. We perform hypothesis testing to identify whether the samples come from a normally distributed population or not. The hypothesis testing is conducted by applying the chi-squared test for normal distribution. The null hypothesis (H_0) is defined as "The population probability distribution is normal". On the other extreme, the alternative hypothesis (H_a) is defined as "The population probability distribution is **not** normal".

We first divide the standard normal distribution $N(0,1)$ into a set A containing eight proportionally equal parts, with each part being equal to 1/8; and then we find a set B containing eight parts in a one-to-one correspondence with the ones belonging to set A such that each sample is assigned onto one of the parts belonging to B. Next, we find how many (estimated) points belong to each of the parts of A. For the first part of A, we find that it contains $np_1 = 131 * 1/8 = 16.375$ (estimated) points, with p_1 representing the probability of first part. Because all the parts have the same probability, we conclude that each part contains 16.375 (estimated) points. We evaluate whether the sample distribution follows the normal distribution through Eq 5 which asymptotically approaches chi-squared distribution X^2. Specifically, E_i represents the estimated points that must belong to the i-th part of B according to the normal distribution. On the other hand, O_i represents the samples (observed points) belonging to i-th part of B.

$$\tilde{X^2} = \sum_{i=1}^{8} \frac{(O_i - E_i)^2}{E_i} \quad (5) \qquad \bar{x} \pm t_{a/2} * \frac{s}{\sqrt{n}} \quad (6) \qquad \sigma_l = s * \sqrt{\frac{n-1}{X^2_{a/2}}} \quad (7)$$

By choosing significance level equal to 0.05, we find through the chi-squared distribution table that the critical region is the region beyond $X^2_{0.5} = 11.07$. Because $\tilde{X^2} = 8.42$, we cannot reject the null hypothesis and we can safely assume that the population follows the normal distribution.

Step 2. We calculate the lower and upper bounds of population mean with 99% degree of confidence, which is a common value. Because the population follows the normal distribution and the deviation σ is not known, we use Eq 6 to calculate the bounds of population mean, where we find that $\mu_l = 115.96$,and $\mu_u = 116.94$.

Step 3. The lower and upper bounds of population standard deviation are expressed by Eq 7 and Eq 8, respectively. By looking into the chi-squared distribution table we find that $\sigma_l = 1.88$ and $\sigma_u = 2.59$.

$$\sigma_u = s * \sqrt{\frac{n-1}{X^2_{1-a/2}}} \quad (8) \qquad Pr(|X - \mu| \geq k\sigma) \leq 1/k^2 \quad (9)$$

Step 4. From Chebyshev's inequality (Eq 9), for $k = 3$ we have that $Pr(X \leq 108.19) \leq 0.11$ and $Pr(X \geq 124.71) \leq 0.11$. As a result, the lower bound of population is equal to 108.9 with 87% (0.99∗0.99∗0.89) degree of confidence, while the upper bound of the population is equal to 124.71 with 87% degree of confidence. Note that we can increase the degree of confidence (by increasing k) at the cost of decreasing/increasing the lower/upper bound of population.

6 Conclusion

In this paper, we propose to design energy-aware nodes of a parallel database cluster to ensure a low energy consumption of the whole platform. Due to the complexity of the

DBMS deployed in a given node, we propose a green-query optimizer build on the top of PostgreSQL. Before building it, an audit has been performed to identify energy-sensitive components of the query optimizers. Based on this audit, a methodology of building such a query optimizer is given. It is supported by an open source tool available at the forge of our laboratory to allow researchers, industrials, and students to get benefit from it. Intensive experiments were conducted to demonstrate the efficiency and usage of our proposal. The obtained results are encouraging. Based on these results a probabilistic proof is given to evaluate the confidence bounds of our model and results. We can conclude that our proposal is complete since it covers a comprehensive methodology supported by an open source tool and a solid mathematical proof. Currently, we are integrating the communication cost in our cost models.

References

1. e Sustainability Initiative, G., the Boston Consulting Group, I.: Gesi smarter 2020: The role of ict in driving a sustainable future. Press Release (December 2012)
2. Poess, M., Nambiar, R.O.: Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. PVLDB **1**(2) (2008) 1229–1240
3. Abadi, D., Agrawal, R., Ailamaki, A., Balazinska, M., Bernstein, P.A., Carey, M.J., Chaudhuri, S., Dean, J., Doan, A., Franklin, M.J., et al.: The beckman report on database research. Communications of the ACM **59**(2) (2016) 92–99
4. Lang, W., Harizopoulos, S., Patel, J.M., Shah, M.A., Tsirogiannis, D.: Towards energy-efficient database cluster design. PVLDB **5**(11) (2012) 1684–1695
5. Li, X., Zhao, Y., Li, Y., Ju, L., Jia, Z.: An improved energy-efficient scheduling for precedence constrained tasks in multiprocessor clusters. In: ICA3PP. (2014) 323–337
6. Boukorca, A., Bellatreche, L., Benkrid, S.: HYPAD: hyper-graph-driven approach for parallel data warehouse design. In: ICA3PP. (2015) 770–783
7. Do, J., Kee, Y.S., et al.: Query processing on smart ssds: opportunities and challenges. In: ACM SIGMOD. (2013) 1221–1230
8. Xu, Z., Tu, Y.C., Wang, X.: Dynamic energy estimation of query plans in database systems. In: ICDCS. (2013) 83–92
9. Xu, Z., Tu, Y.C., Wang, X.: Exploring power-performance tradeoffs in database systems. In: ICDE. (2010) 485–496
10. Kunjir, M., Birwa, P.K., Haritsa, J.R.: Peak power plays in database engines. In: EDBT, ACM (2012) 444–455
11. Lang, W., Kandhan, R., Patel, J.M.: Rethinking query processing for energy efficiency: Slowing down to win the race. IEEE Data Eng. Bull. **34**(1) (2011) 12–23
12. Rodriguez-Martinez, M., Valdivia, H., Seguel, J., Greer, M.: Estimating power/energy consumption in database servers. Procedia Computer Science **6** (2011) 112–117
13. Roukh, A., Bellatreche, L.: Eco-processing of olap complex queries. In: DaWaK. (2015) 229–242
14. Lang, W., Patel, J.: Towards eco-friendly database management systems. arXiv preprint arXiv:0909.1767 (2009)
15. Roukh, A., Bellatreche, L., Boukorca, A., Bouarar, S.: Eco-dmw: Eco-design methodology for data warehouses. In: DOLAP, ACM (2015) 1–10
16. Chaudhuri, S., Narasayya, V., Ramamurthy, R.: Estimating progress of execution for sql queries. In: ACM SIGMOD, ACM (2004) 803–814
17. McCullough, J.C., Agarwal, Y., et al.: Evaluating the effectiveness of model-based power characterization. In: USENIX Annual Technical Conf. (2011)
18. Lisnianski, A., Frenkel, I., Ding, Y.: Multi-state System Reliability Analysis and Optimization for Engineers and Industrial Managers. Springer (2010)