PET: Reducing Database Energy Cost via Query Optimization

Zichen Xu
The Ohio State University
xuz@ece.osu.edu

Yi-Cheng Tu
The University of South Florida
ytu@cse.usf.edu

Xiaorui Wang
The Ohio State University
xwang@ece.osu.edu

ABSTRACT

Energy conservation is a growing important issue in designing modern database management system (DBMS). This requires a deep thinking about the tradeoffs between energy and performance. Despite the significant amount of efforts at the hardware level to make the major components consume less energy, we argue for a revisit of the DBMS query processing mechanism to identify and harvest the potential of energy saving. However, the state-of-art architecture of DBMS does not take energy usage into consideration in its design. A major challenge in developing an energy-aware DBMS is to design and implement a cost-based query optimizer that evaluates query plans by both performance and energy costs. By following such a strategy, our previous work revealed the fact that energy-efficient query plans do not necessarily have the shortest processing time.

This demo proposal introduces PET – an energy-aware query optimization framework that is built as a part of the PostgreSQL kernel. PET, via its power cost estimation module and plan evaluation model, enables the database system to run under a DBA-specified energy/performance tradeoff level. PET contains a power cost estimator that can accurately estimate the power cost of query plans at compile time, and a query evaluation engine that the DBA could configure key PET parameters towards the desired tradeoff. The software to be demonstrated will also include workload engine for producing large quantities of queries and data sets. Our demonstration will show how PET functions via a comprehensive set of views from its graphical user interface named *PET Viewer*. Through such interfaces, a user can achieve a good understanding of the energy-related query optimization and cost-based plan generation. Users are also allowed to interact with PET to experience the different energy/performance tradeoffs by changing PET and workload parameters at query runtime.

1. INTRODUCTION

The fast spreading of energy management research in the field of database management systems (DBMSs) is driven by rapid increase of energy cost in database services [3, 5]. Techniques such as hardware power-model control and query rescheduling [2] have been proposed to save energy in database systems. In comparison to [2], our earlier work [6] presents a framework inside a DBMS kernel for energy reduction while ensuring acceptable performance. In this demo, we will present a more advanced version of that framework named Power-Energy-Time (PET) framework and show its functionalities using an extensive set of database workloads generated from various TPC and scientific database benchmarks. The main purpose of developing PET is to reveal the existence of and study the energy-saving potential of many energy-efficient query plans. The database community has had some controversial views [4] on whether most (time) efficient plans always have high energy efficiency such that energy-aware query optimization is not worthwhile for energy reduction purposes. As our expression to this problem, this demo will show that, in the plan search space enhanced with energy cost estimation for most queries we tested, we could find energy-efficient candidates that are usually ignored by the conventional query optimizer.

To seize such energy-saving opportunities, PET augments the existing query optimizer of PostgreSQL with energy-related modeling and control functionalities. Specifically, the PET query optimizer evaluates query plans by both the (time) performance and energy consumption. For that purpose, a power modeling module is deployed as part of PET to provide estimations of power consumption of query plans of interest. The performance cost (provided by existing query optimizer) and power estimation are treated as inputs to a composite cost model that, in turn, guides query plan selection. Furthermore, a key parameter of the cost model can be used as the handle for the control of query plan selection in order to realize different levels of tradeoff between performance and energy cost. Such tradeoffs are determined by the database service providers and can be specified by DBAs. Due to the dynamical feature of workload and system environment for databases, the capability to adjust such tradeoffs at runtime is essential in the design and implementation of energy-aware DBMS.

For each query sent into DBMS server, the SQL statement will be compiled into a sequence of operations that correspond to relational operators. For each operation, query optimizer decides which relational operator processing algorithm should be assigned to it. For example, for a single table scan, it could be implemented with sequential scan, index scan or bitmap scan in PostgreSQL. Similar decisions are made for joins as well. Those operators, along with the original and intermediate tables, will form a path tree as the implementation for the sequence of operations. In this way, the SQL statement will be executed by processing along the path of this sequence of relational operator. One example of such a query plan tree is shown in Fig. 1. Note that in each node, the cost involves both time and power – this is the main difference between such plan trees in PET and those in traditional DBMSs.

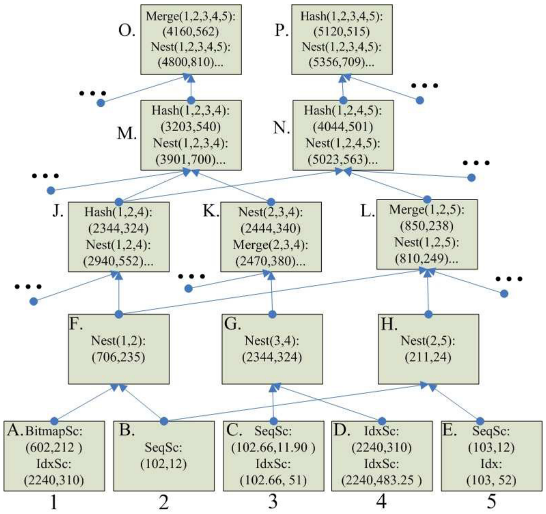


Figure 1: An example of a plan-node path tree of a SQL statement (i.e., Q_2 of the TPC-H benchmark). Each node stands for a relational operator. It contains the possible algorithms for processing this operator and the estimated time and power costs of individual algorithms. The cheapest root node, which is the final execution plan will be selected by query optimizer for processing the query.

Generally, the search space of path tree grows exponentially with the initial number of lead nodes (i.e., number of tables involved in the query). The query optimizer will eliminate most of intermediate nodes since their costs are high and could never been selected as the candidates for the next level node generation based on some predefined rules. This strategy effectively prunes the search space therefore ensures the high performance of the optimizer. To find the cheapest node, the (time) performance cost is estimated before the execution according to the cost model. In producing energy-efficient query path tree, PET follows the same mechanism using a query cost model that considers both energy and performance.

To automate the above process, we design and develop PET in the kernel of PostgreSQL, an open source DBMS.[1] Its kernel includes a power cost estimation model and a optimization evaluation model, which provides functions in the backend of PostgreSQL to generate different path trees for arbitrary queries. To better illustrate the internal mechanisms of the PET framework, we create an graphical user interface called *PET Viewer* associated with our powerful workload generation engine which provides a massive number of workload types for testing the PET functionalities. Another important purpose is to illustrate that there exist energy efficient processing paths in DBMSs, and such paths are not necessarily the ones with lowest performance cost.

2. THE PET FRAMEWORK

This section mainly discusses the important components and functionalities of the PET. In PET, three are three main parts, namely the power cost model, plan evaluation model, and the GUI. Another important part of our demonstrated software, although not a part of the PET framework, is a workload engine. The software architecture of the demonstration can be found in Fig. 2. The workload engine will produce queries according to user's specifications on query arrival rate and query type. When the query enters the system, it will be estimated with its time and power performance cost, evaluated with the customized optimization goal. As a result, a new path tree will be generated as in Fig. 1. When the query is in execution, the power monitor will report the difference of the real power and model estimated power. The difference will be used as a signal to recalibrate the cost model in the DBMS. In this process, PET Viewer serves as a portal that helps the DBA monitor and manipulate the process. More details will be introduced in the following sections.

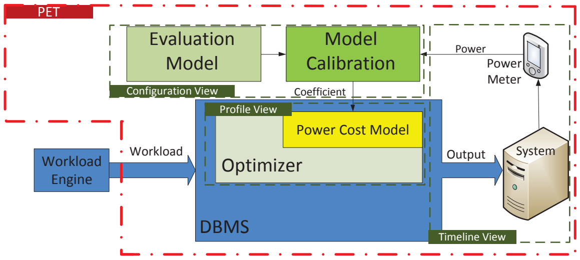


Figure 2: An overview of the PET framework in the demo system, with its different components associated with the three different views of the PET Viewer.

2.1 Power Cost Estimation

Considering a query job Q_i, the power cost P for this query can be represented as:

$$P = W_{cpu} \times N_{tuples} \tag{1}$$

Where W_{cpu} is the power cost weight to process one tuple in CPU and N_{tuples} is the number of fetched tuples from storage.

The theoretical model shown in Eq. (1) is too general to capture the resource consumption patterns of query plans.Specifically, since the cost of different types of operations is different, keeping a single unit power cost parameter is obviously an oversimplified solution. Motivated by time cost estimation models in traditional query optimizers, we could build the power cost estimation model for each relational operator to refine the model in Eq. (1). Note that the query optimizer in a typical DBMS (i.e., those with the System R style design) takes a bottom-up approach to build query trees, and alternative plans in each substree (representing an operator as shown in Fig. 1) will have to be evaluated. With the operator-level cost models, we can follow the same bottom-up strategy to build the cost model for the entire query plan.

As shown in [6], since we have identified CPU as the major active power consumer, we only need to focus on the number of tuples to be processed by the CPU. Each operator could be assigned with one or more power coefficients to estimate its run time power cost. In PET, we deploy power models for a set of popular relational operators. A summary of such operator-level power models can be found in Table 1 while relevant model coefficients are listed in Table 2. As compared to similar models presented in [6], the models listed here in this paper are the results of significant research efforts in power modeling in databases.

The power cost of a plan is calculated from those of the higher-level operations, which consist of basic operations shown in Table 1. Table 2 lists all the important variables for the computation of the power cost for each operators. Clearly, these formulae use the values of basic operations as building blocks. Again, we follow the exact same mechanism for calculating time costs in PostgreSQL to generate these formulae. Readers who are interested in classic time cost estimation in DBMS can refer to the textbook [1].

[1] http://postgresql.org

Table 1: Power cost functions for relational operators.

Methods	Cost function
Sequential Scan	$w_s n$
Index Scan	$w_i n$
Sorting	$w_t nR$
Bitmap Scan	$w_i n + w_t nR$
Nested Loop Join	$w_i(n_1 + n_1 n_2 c)$
Sort Merge Join	$w_t(n_1 R_1 + n_2 R_2) + w_i(n_1 + n_2)$
Hash Join	$w_i\left(\frac{n_1}{H} + n_2\right)$

Table 2: Key quantities in power estimation models.

Symbol	Definition
n	The number of tuples retrieved for CPU processing
w_s	The CPU power cost for processing a regular tuple
w_i	The CPU power cost for processing an index tuple
w_t	The CPU power cost for sorting a tuple
H	The number of hash partitions
R	The number of runs in a sorting algorithm
c	Selectivity of join condition

2.2 Plan Evaluation Model

The plan evaluation model is used to grant the superiority of alternative query plans towards a predefined optimization goal. The conventional cost model in DBMS (i.e. PostgreSQL) only involves the time cost. However, it is no longer the case in PET since we have two performance metrics, namely time and power. In this study, we need a criterion to reflect adjustable trade off between power cost P and processing time T. Specifically, a metric model of the following format is adapted in PET:

$$C = PT^n = ET^{n-1} \tag{2}$$

where C is the aggregated cost, n is a coefficient that reflects the relative importance of P and T, and $E = PT$ is the total energy cost of the plan. Intuitively, it means if sacrificing a 1% degradation in time performance, there would be saving in $n\%$ power. The model is flexible in that it can be used for different optimization goals with the choice of n. More details can be found in [6]. In PET, we will make the model parameter n a runtime knob that can be changed via the PET Viewer. This enables the DBA to adjust system behaviour towards different tradeoffs between performance and energy according to his/her desire (which may involve non-technical factors in decision making).

3. DEMONSTRATION

3.1 Testbed and Workloads

The testbed contains one computer and a power meter (WattsUp-Pro power meter with a $\pm 1.5\%$ measurement error). The laptop, named *server* hereafter, is installed with the PET-augmented PostgreSQL to run DBMS service.

The workload generator produces datasets and workloads that create scenarios of a real-world database services. First, the workload generator borrows data and queries from three sets of benchmarks:

(1) The generator produces a query pool that consists of 2,000 queries derived from the 22 standard queries in the TPC-H benchmark by changing the selection predicates. The workload generator draws queries from such pool with a predefined distribution of query arrival time and features such as the level of resource sharing, query priority, and multiprogramming level.

(2) We demonstrate the PET's capability of energy saving in processing large datasets, we also use a 1TB astronomical database that includes 53 million unique astronomical objects such as stars, galaxies, and quasars. The set of 400 queries against this database are extracted from the query templates posted on the website of the SDSS project – a large-scale scientific database. [2] The SDSS query set mainly consists of large table scans and joins of few tables (mostly two-table joins).

(3) Finally, we use a TPC-C benchmark tool named TPCC-UVa[3] to generate OLTP-style workloads. One thing to point out here is that TPCC-UVa is a closed benchmark tool in that users cannot access or modify the queries. Such a tool forms a black-box testing environment for the effectiveness of the PET functionalities.

3.2 The PET Interfaces

The demonstration will use the *PET Viewer*, a new graphic user interface we built for better illustrations of the PET runtime environment. Specifically, users can use this GUI to:

(1) get better on-the-fly observations of the details of query execution in a real DBMS such as the query plan trees;

(2) provide customized optimization goals to achieve desirable energy/performance tradeoffs in the DBMS query processing engine; and

(3) track the energy consumption of different database workloads at runtime for the purpose of energy benchmarking.

For the above purposes, we summarize the core functionalities of PET Viewer into those related to the power cost model, evaluation model and real-time execution. For each category of functionalities, we provide a corresponding interface (view) as follows:

- *Configuration View*, used to define the optimization goal and concurrency degree.
- *Profile View*, used to display the detailed information of running query, such as execution path tree and its energy and performance cost.
- *Timeline View*, used to visualize the run time power usage of query execution and other energy-related statistics.

3.2.1 Configuration View

Our demonstration suggests a set of queries covering all standard operations of our power profiling experiments, from small transactional operations to more complex aggregates on larger sets of data. Any users are allowed to modify the specifications of any query (e.g., the attribute that affects the search selection). Some samples of the default queries for our demonstration are as follows:

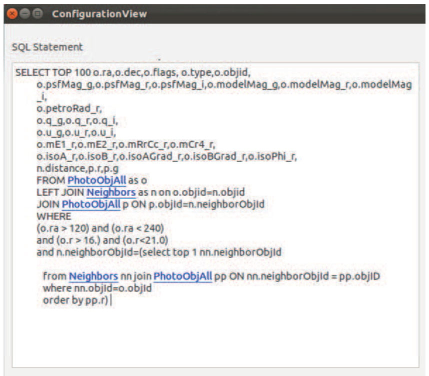
Q_1 Search an item in the lineitem table:
SELECT l_name FROM lineitem WHERE ...

Q_2 Find all information for a shipped item:
SELECT p_mfgr, s_address, s_phone, ...
FROM part, supplier, partsupp,nation, region
WHERE ...

Q_3 Return 1000 objects in Galaxy table:
SELECT TOP 1000 objID
FROM Galaxy
WHERE ...

[2] http://www.sdss.org/dr7

[3] http://www.infor.uva.es/˜diego/tpcc-uva.html

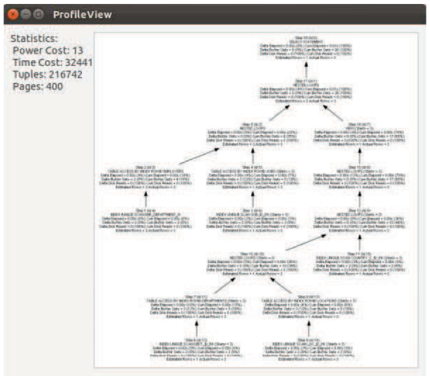


SQL Statement

```
SELECT TOP 100 o.ra,o.dec,o.flags, o.type,o.objid,
      o.psfMag_g,o.psfMag_r,o.psfMag_i,o.modelMag_g,o.modelMag_r,o.modelMag
      _i,
      o.petroRad_r,
      o.q_g,o.q_r,o.q_i,
      o.u_g,o.u_r,o.u_i,
      o.mE1_r,o.mE2_r,o.mRrCc_r,o.mCr4_r,
      o.isoA_r,o.isoB_r,o.isoAGrad_r,o.isoBGrad_r,o.isoPhi_r,
      n.distance,p.r,p.g
      FROM PhotoObjAll as o
      LEFT JOIN Neighbors as n on o.objid=n.objid
      JOIN PhotoObjAll p ON p.objId=n.neighborObjId
      WHERE
      (o.ra > 120) and (o.ra < 240)
      and (o.r > 16.) and (o.r<21.0)
      and n.neighborObjId=(select top 1 nn.neighborObjId

        from Neighbors nn join PhotoObjAll pp ON nn.neighborObjId = pp.objID
        where nn.objId=o.objId
        order by pp.r)
```

Figure 3: An query editor interface part of the Configuration View of PET Viewer.



Statistics:
Power Cost: 13
Time Cost: 32441
Tuples: 216742
Pages: 400

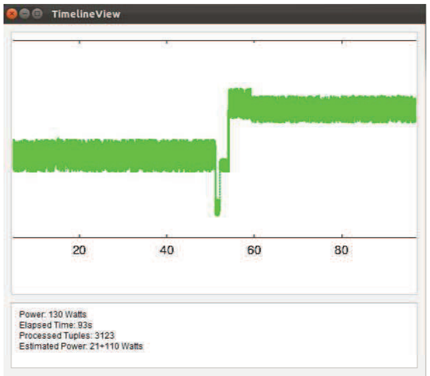
Figure 4: The profile view interface of the PET Viewer showing details of a benchmark query.

Q_4 Find galaxies in a given area of the sky:
SELECT colc_g, colc_r
FROM Galaxy
WHERE ...

Q_1 and Q_2 are extracted from TPC-H standard benchmark while Q_3 and Q_4 are borrowed from SDSS sample queries of exploring the SDSS astronomical database. The search range and attribute can be modified in the configuration view. Also, it allows users to get a specific range so that the workload engine will automatically generate multiple copies of the query with different projection statements. A snapshot of the interface can be found in Fig. 3.

3.2.2 *Profile View*

Once the query is fed into the DBMS engine and the query optimizer computes the optimal execution plan, the path tree that we discussed above will be printed out as the reference to understand the processing of this query. We will show that when different optimization goals are given by the configuration view, different path trees of the target query can be displayed in the profile view. Also, we will allow users to give runtime hints to affect query optimizer in plan selection. For example, we can enforce PET to use a particular type of operator processing algorithm (e.g., hash join, sequential scan). A snapshot of the profile view is shown in Fig. 4. In summary, it assists users to read the runtime estimation cost to understand how the cheapest performance plan or power-oriented plan are generated, whether they are different and why.



Power: 130 Watts
Elapsed Time: 93s
Processed Tuples: 3123
Estimated Power: 21+110 Watts

Figure 5: The runtime statistics of executing one query in PET is shown in the Timeline View.

3.2.3 *Timeline View*

When the query is in execution, we will show the runtime power cost of the whole machine and all the statistic data from previous profile view. A snapshot of the Timeline View is shown in Fig. 5, which displays the statistics of the running query. Also, via the different plans chosen according to the different optimization goal, we could observe real power savings by only manipulating the internal processing structure of the DBMS instead of installing new hardware devices.

4. SUMMARY

This demo will serve the purpose of illustrating the extensive experiments and conceptual explorations we conducted in the topic of energy-aware database systems. With those experiments and explorations, we could firmly stand behind our argument that power-aware DBMS is not only a hardware issue or operating system issue. Therefore, we need to rethink the design of the DBMS to allow different tradeoffs between energy and performance metrics. In this way, we may not optimize the whole system towards best time performance. Instead, the goal is to build a balanced system that best fits the performance and economical/environmental requirements of the database service provider.

5. REFERENCES

[1] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, 2008.

[2] W. Lang and J. M. Patel. Towards eco-friendly database management systems. In *CIDR*, 2009.

[3] 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.

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

[5] U.S. Environmental Protection Agency. Report to congress on server and data center energy efficiency. *Government Report*, 2007.

[6] Z. Xu, Y.-C. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *ICDE*, pages 485–496, 2010.