Online Energy Estimation of Relational Operations in Database Systems

Zichen Xu [#], Yi-Cheng Tu [*], Xiaorui Wang [#]

[#]The Ohio State University, [*]University of South Florida

Abstract—Data centers are well known to consume a large amount of energy. As databases are one of the major applications in a data center, building energy-aware database systems has become an active research topic recently. The quantification of the energy cost of database systems is an important task in design. In this paper, we report our recent efforts on this issue, with a focus on the energy cost estimation of query plans during query optimization. We start from building a series of physical models for energy estimation of individual relational operators based on their resource consumption patterns. As the execution of a query plan is a combination of multiple relational operators, we use the physical models as a basis for a comprehensive energy model for the entire query. To address the challenge of maintaining accuracy under system and workload dynamics, we develop an online scheme that dynamically adjusts model parameters based on statistical signal modeling. Our models are implemented in a real database management system and evaluated on a physical test bed. The results show that our solution achieves a high accuracy (worst-case error 13.7%) despite noises. Our models also help identify query plans with significantly higher energy efficiency.

Index Terms—Energy-aware Database System Design, Energy Estimation, Online Estimation, Query Optimization

1 Introduction

Data centers (DCs) are known to be the "SUVs of the tech world" for their enormous energy consumption. Triggered by this phenomenon, recently there are a lot of efforts on energy management in data centers [1], [2], [3], [4], [5], [6], [7], [8]. However, by focusing on the operating system (OS) level, such work cannot be directly applied to application-level energy management, due to the lack of sufficient knowledge of the application behavior. In this paper, we study energy management mechanisms in a very important type of DC application – database management systems (DBMSs). The energy reduction in DBMSs are of high economical significance for DCs. In a typical DC, database servers consume the majority of the computing resources, making DBMS the largest application consumer of energy. For example, [9] reported a power consumption ratio of 11 : 9.9 for the back-end DBMS services to the front-end web services.

Energy management has become an active research topic in the database research community. The main theme in such endeavors is to design DBMSs with energy consumption as a first-class performance goal, as advocated by the Claremont report [10]. Current work in energy-aware DBMS has focused on energy-aware query optimization that considers both processing time and energy consumption ([11], [12]), and power management policies in distributed databases ([4], [13]). Unlike other studies that focus on the implementation of energy-aware DBMS, this paper addresses on a key issue that has so far received little attention – modeling the energy cost of database systems. In particular, we report the results of our study in *energy cost estimation in DBMS during query optimization, with a focus on the quantification of the energy consumption of query plans*.

Energy cost in database operations carries high technical significance in energy-aware database design. In database systems, the query optimizer evaluates different computational paths (named *plans*, as shown in Figure 1) by explicitly labeling their resource consumption. The knowledge of the energy consumption of query plans is indispensable in identifying those with a low energy profile [14]. For example, recent studies [11], [12] have shown that in a typical database, there exist many query plans that consume much less energy with little performance degradation. Thus, energy conservation can be achieved by selecting those query plans. Note that information needed for making the decision is hidden inside the database system, thus cannot be captured at the OS or the hardware level. To harness the energy saving opportunities provided by energy-efficient plans, a practical approach is to provide accurate energy cost estimation during the query optimization process.

Energy cost estimation in the DBMS serves two purposes. First, like the traditional cost estimation mechanism in the DBMS that helps select faster query plans, energy cost estimation enables selection of query plans with lower energy cost [12]. Second, knowing the accurate energy cost of each selected query plan helps quantify the energy cost of the entire workload. Therefore, we believe our work is important for query optimization design in an energy-aware DBMS, and it also provides valuable insights for other energy management policies, such as energy consolidation and projection in DCs [3].

Specifically, we design and evaluate a two-level framework to fulfill the above design goals. In a DBMS, each query plan is a unique path to execute a series of re-

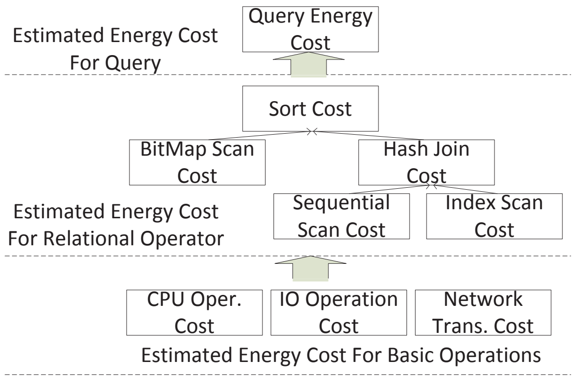

Fig. 1. The query plan of Q5 from the TPC-H benchmark [15]

lational operators (Figure 1) and maintenance operations (e.g., update/delete), each of which encompasses a set of basic operations (e.g., processing a tuple in CPU). We first introduce our study on the energy consumption of each basic operation from the energy profile of hardware in a typical database server. Based on which, we build a static model that describes the energy consumption of relational operators and maintenance operations according to their estimated resource demand (i.e., the CPU cycles). Those resource data are provided by the traditional DBMS optimizer and their energy cost coefficients are derived from a training process. Such models show a high accuracy in predicting energy consumption in a static environment. However, the energy cost coefficients (e.g., number of joules needed to process an indexed tuple) in the model depend on the system dynamic (e.g., CPU utilization) and the workload statistic (e.g., table cardinality). To further improve the static model by making it adaptable, we propose an online scheme that uses a Recursive Least Square (RLS) estimator to periodically update the parameters in the static models.

The problem we tackle is to build a comprehensive model that can predict the energy cost of query plans in a DBMS. To be more specific, given one query, our goal is to *accurately quantify the active energy consumption if that plan is executed*. The desired energy cost model should possess the following features:

- *Accuracy*: the model provides accurate prediction of energy consumption;
- *Robustness*: the model maintains high accuracy regardless of variations from system states and workload characteristics;
- *Fast Response*: the computational overhead is acceptable;
- *Non-disruptive*: the modeling process does not interfere with the normal operations of the DBMS.

Among the four, *accuracy* and *robustness* are the key requirements, and thus the main metrics for evaluating our models in this paper.

1.1 Contributions and Paper Organization

Previous studies on database energy management have focused on either high-level ideas [11] or energy profiling of hardware used in data processing [16]. In this paper, we introduce a systematic solution with all aforementioned properties. The main technical contributions of this paper are:

- We propose a comprehensive mechanism to estimate the energy cost of query processing at the DBMS level and develop a general energy model based on an extensive system identification study.
- We develop a series of physical models based on the general model for evaluating the energy cost of individual relational operators in a static environment.
- We design an online scheme to automatically adjust parameters of the static model in response to the system dynamics and the workload variations;
- We implement our model in the kernel of a real DBMS, and evaluate it on a physical testbed with a comprehensive set of workloads generated from TPC benchmarks and scientific database traces.

The remainder of this paper is organized as follows: we first compare our work with other projects in Section 2. We then provide our system identification studies in Section 3. We describe the technical details of our static energy estimation models in Section 4, evaluate those models, and discuss their limitations in Section 5. We present the online scheme to improve the applicability of the model and evaluate it with many experiments in Section 6. We study the impact of the energy efficient query optimization in Section 7. Finally, we conclude the paper in Section 8.

2 Related Work

Energy modeling in operating systems: there are many proposals that treat energy as a first class resource in the operating system, such as [17], [18], [19]. The first formal analytic energy model in the operating system is proposed by Heath *et al*. [20]. Following the similar idea, many other articles reported various models towards energy optimization, such as [21], [22], [23], [24], [25]. Comparing with our work, all such solutions focus on the operating system level. As a result, none of the models can be directly applied to DBMSs, due to the lack of knowledge of the DBMS resource demand and the data processing patterns.

Energy management in database systems: work in energy-efficient database systems can be traced back to the early 1990's. In [26], query optimization with energy as one of the performance criterion was proposed within the context of mobile databases. In this paper, we are interested in the energy consumption of servers. Motivated by the increasing energy-related cost of database servers, the database community has only recently identified building energy-efficient database systems as an worthy direction of exploration [10]. Early papers [27], [28] advocated query optimization with energy as the target, which implicitly argue for a mechanism for estimating the energy cost of a query plan. Supported by initial experimental results, [14] presented two specific techniques to save energy in databases: decreasing CPU frequency and throttling user queries. Our previous work [12] revealed the existence of many energy-efficient query plans that carry little/acceptable performance penalty. By showing some plans of high energy efficiency coincides with performance, a subsequent report [29] stirred up discussions on whether energy-aware query optimization is a worthwhile approach towards green databases. Our opinion is that, when the search space is sufficiently large and energy/performance estimations are accurate enough, we would find energy-efficient plans that could most likely be ignored by existing query optimizers. This standpoint is supported by more recent evidence provided by [11] and [16], and verified by our experimental results shown in Section 6.2. Other related research in green databases diverges to several directions. The Transaction Performance Council (TPC) announced TPC-Energy [13] in 2007. Poess and Nambiar [9], [30], [31] reported extensive results on power consumption patterns in typical database servers.

Modeling power/energy in databases: it is worth noticing that power/energy modeling has been addressed in some of the work mentioned above. As a position paper, [11] proposed a general formula for quantifying power cost of a query plan. Another work [16] delivered more comprehensive results in modeling the peak power of database operations. As the peak power and the energy are very different concepts, the modeling processes (and apparently the models) are also different. A shorter version of this paper [32] focuses on building physical models on energy consumption and significantly improves the static models from [12]. Aiming at a robust solution with high accuracy in realistic relational database environment, we use a dynamic modeling approach to continuously update key parameters of our model so that it adapts quickly to system dynamics and workload variations. Compared to [32], this paper provides a broader discussion on the energy profile of important relational operations and many database maintenance operations. We also present our study on the energy efficiency of popular join algorithms based on our understanding of their energy consumption patterns. Furthermore, we highlight our online solution with a study on how accurate energy estimation helps save more energy by altering query plan selection, and share insights on query optimization towards low energy cost.

Other related work: there are numerous reports on dynamic power management (DPM) at the operating system level, and many DPM techniques are summarized in the survey [3]. Cost modeling of relational operators is a well-studied problem in the database field. Work related to this topic can be traced back to the late 1970's. Initially, Astrahan and co-workers presented some critical ideas implemented in System R [33]. Christodoulakis [34] summarized the early work and well-accepted assumptions for query cost estimation. In [35], the authors extend the work to a distributed environment. Standing on their shoulders, we build up our physical models based on similar assumptions and techniques. However, as we attempt to model a different cost, variants and constraints are no longer the same.

3 Overview of Modeling Process

In a traditional DBMS [35], query execution cost is treated as a linear combination of three components: CPU cost, I/O cost, and communication cost. Such costs are normally measured as the product of the number of basic operations in the query plan and the resource consumption of each basic operation. The operations involved are: number of tuples (N_{tuples}) to be processed in the CPU, number of pages to be retrieved from disks (N_{pages}), and the number of bytes to be transmitted via networking system (N_{msg}). With the derived energy coefficients from experiments, an intuitive model can be built to describe the energy consumption of the query plan. This heuristic model follows the intuition of treating energy as a resource consumed by the internal processing in the DBMS, and can be treated as the starting point of our work in energy cost estimation. Specifically, the energy cost $\hat{E}$ of a query plan can be expressed as

$$\hat{E} = W_{cpu}N_{tuples} + W_{I/O}N_{pages} + W_{msg}N_{msg} \tag{1}$$

where W_{cpu}, $W_{I/O}$ and W_{msg} are the energy coefficients of one tuple processed in CPU, one page obtained from disks and one byte processed in network, respectively. Our work focuses on modeling energy consumption of processing relational operators on a single-node database server, therefore we do not consider the network transmission cost in this paper. However, our model could be easily merged with other state-of-art network power models, such as those described in [36] and [37], to provide an overall energy profile in a distributed database system. As a result, the above model becomes:

$$\hat{E} = W_{cpu}N_{tuples} + W_{I/O}N_{pages} \tag{2}$$

As a general linear model, parameters of Equation (2) are obtained by a series of *system identification studies* on energy consumption of hardware components and typical database workloads. In the experiments, we build a set of queries, each of which performs a single table scan or a two-table join, to examine the energy profile of each relational operator. At the same time, the query optimizer is modified to only pick the algorithms we specify for processing a relational operator. We pick the scaling factor of TPC-H tool from 1 to 1000, which leads to 1GB to 1TB data stored in our local drive. The system identification study picks the number of tuples as the metric for the data size. Each tuple represents a row in the data table. Queries are based on the TPC-H query generator using the standard 22 query templates. We collect the runtime statistics (e.g., resource consumption) and energy consumption data of most relational operators at a frequency of 1000Hz using the multi-meter and the Linux system monitor. Based on those data, we calculate the coefficients of the physical models using the solvers from GAMS[1]. More details of our experimental setup can be found in Section 5.1.

1. http://www.gams.com

TABLE 1
The maximum active power consumption of major hardware components in a R710 Dell 2U server.

Component	Active Power (Watt)	Citation
CPU: Xeon E5645	118.9	[29]
Memory: 32 GB	20.5	[21]
HDD: Seag. 2TB 7200RPM	0.42	[30]
Other parts	0.23	N/A
Total	140.05	

3.1 Observations on Hardware

First, we start to explore the energy profile of the hardware, as part of the system identification study. To that purpose, we measure active power consumptions of major hardware components (shown in Table 1) of a database server as our testbed. The results exhibit the fact that CPU and memory contribute the most to the active power (about 99%), as shown in Figure 2. The active power consumption contributed by other components (e.g., hard disk) are negligible. From Figure 2, we found that (1) the CPU power cost is proportional with the workload density, which indicates the system utilization, (2) memory rarely stays idle, thus its energy consumption is only related to the processing time. Meanwhile, as shown in Figure 2b, the disk power is not affected by data access patterns – both sequential and random access consume the same power. This is due to an important physical feature of commonly-used storage hardware for database servers – their leakage power cost always dominates (solid-state drives could be different but they are rarely used for storing large databases).

To estimate the energy cost of a query plan, we are essentially interested in its **marginal energy consumption** (namely active energy) if we assume the baseline power is always the same as the leakage power.[2] Note that, without specification, the energy consumption discussed throughout this paper is the active energy consumption. In the experiment, we have confirmed our intuition that the marginal energy consumption of a query plan is proportional to its resource demand (i.e., number of operations N_{tuples}), which captures the processing time of such query. Thus, to estimate the energy consumption of a database operation, we need to know the total resource demand (e.g., N_{tuples} and N_{pages}), and the energy coefficient parameters, which represent the processing time and the power cost of executing such query plan, respectively.

3.2 Observations on Relational Operators

Scan: We are interested in verifying whether above models hold in different query processing patterns. Therefore, we extend the identification experiments on each relational operator. The results of our extensive experiments using typical database workloads in Figure 3, however, show that the CPU energy consumption does **not** always linearly increase with the number of processing tuples (e.g., Equation (2). In other words, power does not always increase - it levels out beyond a certain value of N_{tuples}. In such experiments, we

2. This assumption will be relaxed in Section 6.

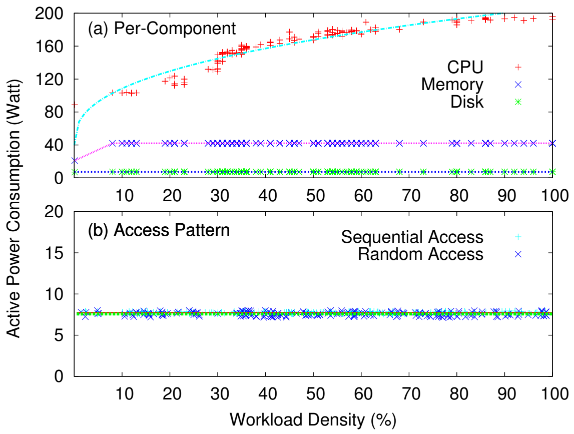

Fig. 2. (a) CPU, memory and hard disk power profiles. (b)Hard disk power consumption under sequential and random reads

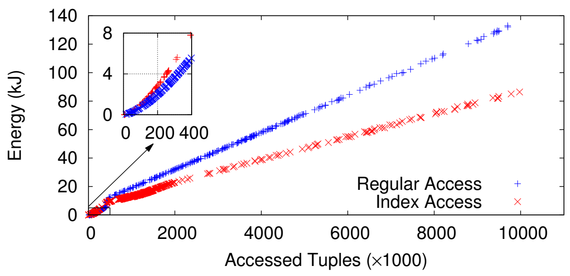

Fig. 3. Active energy consumption under different scan algorithms

run the same query repeatedly in databases with different names and sizes to avoid the impact of resource sharing. Figure 3 shows the CPU energy consumption of two types of queries: one with the sequential scan and the other with the indexed scan. For both queries, we can observe that, the CPU energy consumption first exhibits a non-linear growth with the total number of tuples accessed until reaching its “hockey point”. After this point, the relationship between energy cost and query size becomes linear. By looking deeper into the low-level operations, we believe reasons for the above observations are: when the number of processing tuples is small (i.e., before the hockey point), the energy consumption is dominated by the CPU usage, which has a quadratic growth with the size of the input (shown in Figure 2). When the system is fully utilized, the CPU energy coefficient (i.e., power) is almost a constant, shown as the tail of the curve in Figure 3. Meanwhile, we can also observe that the shape and hockey point of the curve are different in different scan methods. *Therefore, we need to consider models for each individual relational operation.* To represent the piecewise curve in Figure 3, we revised the energy model as:

$$\hat{E} = \begin{cases} W_{cpu} N_{tuples}^{m+1} + \overline{W}_{I/O} N_{pages} & \text{for} \quad N_{tuples} \le N \\ W_{cpu} N_{tuples} + \overline{W}_{I/O} N_{pages} & \text{for} \quad N_{tuples} > N \end{cases} \tag{3}$$

where N is the hockey point of the energy cost curve. Based on the regression curve obtained from system identification experiments, $m \approx 0.5$ in our platform.

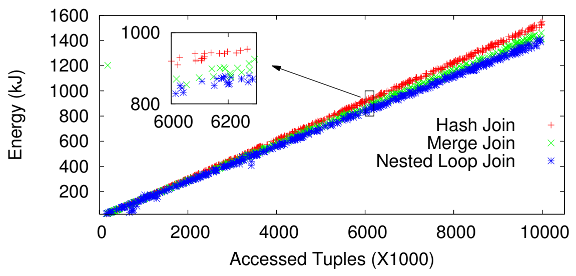

Fig. 4. Energy consumption of join operations using different algorithms

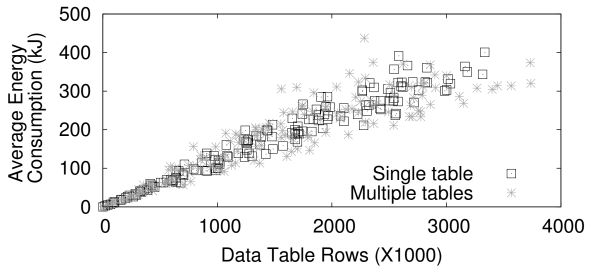

Fig. 6. Energy consumption of update operations on single/multiple table(s) with sizes.

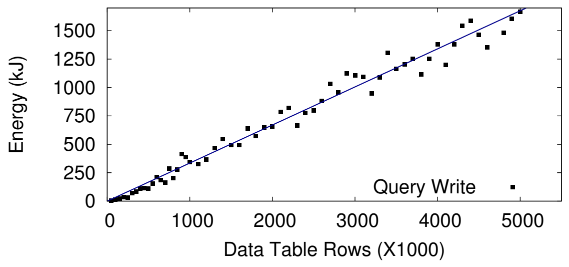

Fig. 5. Energy consumption of the update operation on data tables with different sizes

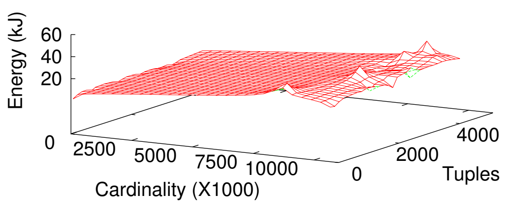

Fig. 7. Energy consumption of deletion operation under different numbers of tuples removed and table cardinality

Join: We also run similar experiments for join-based queries, which contains only one join operation for examining different join algorithms. After eliminating the energy cost of the scan operations (because a join always happens after a scan of the two input tables), we find that the energy cost of the join operation has a linear relationship with the input of the join operation (number of tuples), as illustrated in Figure 4. This input, provided by the optimizer engine in the DBMS kernel, is the estimated size of the temporary table after low-level scans. Furthermore, different join algorithms carry different energy cost. For example, hash joins always consume more energy than the other two joins under the same input. Therefore, to model the linear relationship between one join operation and its energy cost, we are looking for the unique energy coefficient for each join algorithms.

Update and Insert: Update and insert are representing the major operations for write queries in the DBMS. We start examining the energy consumption of the update operations by changing values of records in various data tables with different sizes. All experiments are running in a batch mode to avoid interference from other database operations. As highlighted in Figure 5, the energy consumption of updating records increases monotonically with the size of the affected table. Most of the energy consumption is contributed by the scan to find the victim record(s) to be updated. At the same time, due to repeatedly scanning the same file for finding the victim records and related operations for writing back data, the non-linear behavior from scanning is less observable for update operations. A linear model fits the curve with $R^2 > 93\%$. Therefore, we model the energy cost of update operations as a linear model.

We further investigate the behavior of update operations on multiple table. In a relational database, many data tables are created with foreign keys for cross references. In this case, if one record is updated in a table, the content of multiple tables are updated as well for consistency. We conduct experiments on updating records that may affect single table and multiple tables. We conduct experiments on updating records that affects single and multiple database tables, as shown in Figure 6. The results show that the energy consumption data of updating one record on one table file with 1 million rows and on multiple table files with a total size of 1 million rows are close. Thus, it is safe to only use the number of affected records (rows), or the input size, as the variable in the linear regression model.

Deletion: We consider the deletion operation in two scenarios: (1) removing different number of rows in one data table. The purpose of this scenario is to check the impact of the size of the deletion operation on the energy consumption. (2) removing the same number of rows in two tables with different sizes. We run this experiment with different deletion sizes to confirm our findings in scenario (1). The results are shown in Figure 7. The deletion operation contains a scan operation that finds the victim row(s), and a follow-up operation that marks the data location as invalid and rebuilds indices of the affected tables. According to Figure 7, the energy consumption shows a linear increase with the number of deleted tuples. When the query tries to remove tuples from different files, the size of the affected table files also have an impact on the energy consumption, as illustrated in Figure 3. Thus, there are two parameters in the modeling, (1) the number of affected rows and (2) the size of affected table.

Create and Delete Tables: Detecting the energy consumption of create and delete operations are hard because com-

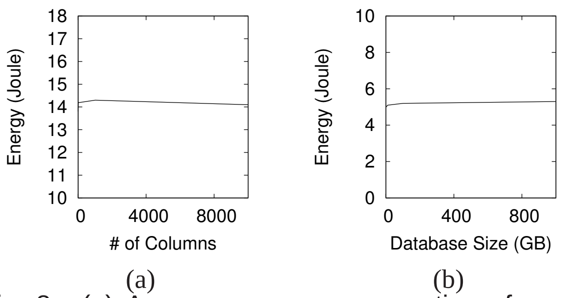

Fig. 8. (a) Average energy consumption of creating databases with different sizes. (b) Average energy consumption of deleting databases with different sizes

paring with other operations, creating/deleting a database or a table is very fast. Therefore, we conduct a script to create a large number of databases and tables. The energy cost is calculated as the average among all runs.

As illustrated in Figure 8(a), the energy consumption of creating a table is almost a constant. It is because the creating operation only affects the hardware once. The number of columns of a table does not affect the energy consumption of creating/deleting a data table.

The cost of the table deletion operation is not correlated with the size of tables (Figure 8(b)). This is because deleting a table only involves marking a certain block of data in the physical storage as invalid. The buffer manager of DBMS would use it for other purposes and the OS overwrites this physical address space afterwards. In all, the energy cost of table deletion operation is constant.

Based on the findings of the above system identification experiments and refined model Equation (3), it is necessary to quantify the number of operations N_{tuples}, the model parameters (i.e., W_{cpu}, $\overline{W}_{I/O}$) and the hockey point N for the table scan operation. $\overline{W}_{I/O}$ is a hardware-specific constant. The quantity N_{tuples} is readily available from the existing query optimizer.

4 Static Modeling

For each operator, the model (Equation (3)) should be modified based on its processing behavior. In the remainder of this section, we introduce energy models for a set of popular relational operators. Readers interested in more detailed work on the model calibration can refer to [38]. A summary of the operator energy models can be found in Table 4 with all the symbols introduced in Table 2. Note here, all the variables in Table 2 except w_x and N_x can be obtained from the existing DBMS optimizer.

4.1 Scan

For single table relational operators (i.e., *selection* and *projection*), we only consider two file organizations – heap files and indexed files, and their corresponding scanning algorithms – *sequential scan* and *index-based scan*, respectively. In addition, we consider a special type of index scan – *bitmap scan* that is implemented in the PostgreSQL system. *Sorting* is a very important step in processing many

TABLE 2
Key quantities in energy estimation models

Symbol	Definition
n	Number of tuples retrieved for CPU usage
p	Number of pages retrieved from storage
R	Sorting algorithm coefficient
x	Indicator of chosen relational operator
w_x	CPU unit-energy cost of relational operator x
$\overline{w}$	Per-page I/O energy cost of relational operator x
N_x	Hockey point of relational operator x
C_x	Constant cost of relational operator x

TABLE 3
Energy cost coefficients for relational operators

Var	Seq	Idx	Sort	Bmap	Nested
w_x	0.0078	0.0093	0.1098	0.0193	0.153
N_x	1153	2109	N/A	2654	N/A
	Merge	Hash	Update	Remove	
w_x	0.165	0.189	0.0027	0.0039	
N_x	N/A	N/A	N/A	N/A	

TABLE 4
Energy cost functions for relational operators

Methods	Cost function
Sequential Scan	$w_s n^{\frac{3}{2}} + \overline{w}p, n \leq N_s$; $w_s n + \overline{w}p, n > N_s$
Index Scan	$w_i n^{\frac{3}{2}} + \overline{w}p, n \leq N_i$; $w_i n + \overline{w}p, n > N_i$
Sorting	$w_t nR$
Bitmap Scan	$w_b n^{\frac{3}{2}} + w_t nR + \overline{w}p, n \leq N_b$; $w_b n + w_t nR + \overline{w}p, n > N_b$
Nested Loop Join	$w_l n + \overline{w}p$
Sort Merge Join	$w_m n + w_t nR + \overline{w}p$
Hash Join	$w_h n + \overline{w}p$
Update	$w_i n + w_u p + \overline{w}p$
Deletion	$w_i n + w_r n + \overline{w}p$

relational operators, thus its energy cost in DBMS is also considered (although it is not a relational operator *per se*).

Sequential Scan: Sequential scan searches each row of the heap file (data table) and omits relevant tuples according to the predicate. The anticipated energy cost, according to Equation 3, is $w_s n^{\frac{3}{2}} + \overline{w}p$, before reaching the hockey point. After that point, the estimated energy cost is $w_s n + \overline{w}p$. Note here, we use n as the estimated number of basic operations (i.e., resource demand) in the model throughout this paper.

Index Scan: Index scan is similar as sequential scan except it uses a (tree-based or hash) index to reduce the number of tuples accessed. The estimated energy cost for index scan is $w_i n^{\frac{3}{2}} + \overline{w}p$ for searching the n anticipated tuples from p pages before the hockey point. When the system is saturated at the hockey point, estimated energy cost is $w_i n + \overline{w}p$. Note here, the unit energy cost of accessing an indexed tuple w_i is different from that of a tuple in sequential scan (w_s).

Sorting: Sorting is a CPU-intensive operation. Although it is not a relational operator *per se*, the sorting operation is often used before/after multiple joins for the optimization purpose. For example, sorting is usually used in the sort merge join, and in the implementation of PostgreSQL, the sorting operation is usually executed after each join no matter which join method is used. The sorting size and the specific sorting algorithm are the key factors in estimating energy cost. The energy cost for sorting is $w_t nR$, where n is the number of tuples fetched to be sorted, w_t is the related energy coefficient, and R is an algorithm-specific coefficient. For the merge sort algorithm implemented in PostgreSQL, we set $R = 29.89$ based on the configuration of maximum rows in a table and linear regression data. The R for other sort algorithms implemented in different database systems may be different and can be derived from identification studies.

Bitmap Scan: Bitmap scan searches the data file using a bitmap index, which is based on bit arrays (i.e., bitmaps) of columns. Then, the scanned result is sorted by the bitmap index. Thus, its energy cost is $w_b n^{\frac{3}{2}} + w_t nR + \overline{w}p$ before the hockey point and $w_b n + w_t nR + \overline{w}p$ for the rest.

4.2 Join

For any two table joins (using original or temporary tables), the energy consumption depends on the join algorithm used. According to Figure 4, the energy consumption grows linearly with the joined data size after eliminating low-level scan costs. Thus, we apply the similar linear model to the nested loop join, the sort-merge join and the hash join.

Nested loop join: Besides the cost of aggregating the n tuples as the output from the low level scans, The energy consumption of the nested loop join is the aggregated cost of the CPU computation from iterating the worst-case n outer loops and related memory accesses. According to the identification study, the energy consumption is linearly correlated with the number of affected tuples (in blocks). Thus, the cost function is $w_l n + \overline{w}p$.

Sort Merge join: The sort merge join includes the sort operation of the sub-lists as the intermediate results of the two lower level nodes. After the sorting, the sort merge join consumes $w_m n$ joule of energy for merging. Meanwhile, we also consider the cost of related memory accesses. The total cost is $w_m n + w_t nR + \overline{w}p$.

Hash join: The cost model of the hash join is similar as the one from the block nested loop join. The cost contains building and probing phase during the computing. Unlike time estimation, for energy consumption, we consider the cost of building the hash table during the probe phase. The hash join identifies the outer and inner table in a subsequent probing phase, and takes hashing to both relations on the join attribute. We generate the cost function due to the energy cost of both phases are proportional to the input size n and related I/O cost from accessing p pages, as $w_h n + \overline{w}p$.

Analysis of Energy Profiles of Join Algorithms. The high cost of joins motivated us to further study the energy profiles of different join algorithms. In particular, we aim at an in-depth understanding of the energy consumption patterns of such algorithms by investigating their hardware use patterns. Hardware counters are the number of low-level hardware activities during the execution of a computing thread. Here we use this tool to learn the runtime hardware behavior of different join algorithms. In our study, we perform over 1000 queries using the TPC-H scheme that have join operations in PostgreSQL (Hardware setup details are in Section 5.1). We record the following performance counters: number of CPU cycles (CYC), (i.e., all used CPU cycles related to the thread), last-level cache misses (TCM), CPU cycles stalled for memory access (SCY). We choose those counters based on the results from [39], which represent the energy performance of CPU and memory for specific processes.

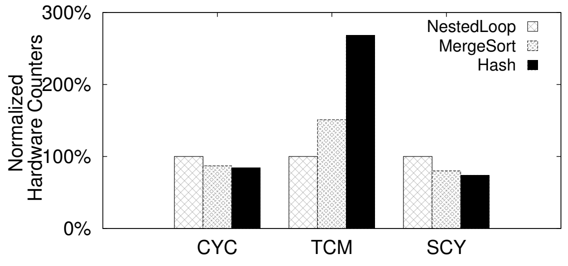

Fig. 9. Normalized performance counters [39] from three joins.

As shown in Figure 9, on average, the nested loop join only uses 15%-17% more CPU cycles. This is surprising since the nested loop join, being a quadratic algorithm, processes much more tuple-to-tuple comparisons than the other two algorithms. Most cycles are stalled for memory operations. Compared with the nested loop join, the hash join has two times more cache misses because it frequently accesses the in-memory hash table. As a result, the hash join has a much higher energy footprint than the nested loop join due to the high power consumption from cache management. On the other hand, the sort merge join seems to be a tradeoff between these two joins. the sort merge join uses slightly more CPU cycles but has much less lower cache misses than the hash join. Based on results from Figure 9, we find that the sort merge join is a safe choice to select for saving energy (more discussions on the selection choices are presented in Section 7).

4.3 Other Operators

Update: update operations write the new data into the row(s) of data table after finding the target row(s). We assume the procedure of finding such target always uses index scan. Therefore, the energy consumption is the combination of the scan cost and update cost. We use a linear model to add those cost up, and it gives $w_i n + w_u p + \overline{w}p$.

Deletion: the deletion operation is similar to the update operation. The only difference lies on the operation after finding the target row(s). The deletion operation marks certain data in memory as invalid and releases the memory space. This is an atomic operation, thus the energy consumption of a deletion operation is $w_i n + w_r p + \overline{w}p$.

5 Model Validation

5.1 Experimental Setup

Our testbed consists of a 2U Dell R710 (3.0 GHz 12-core CPU Xeon E5645, 32GB of DDR3 memory, and 2TB

7200RPM HDD as local storage, as shown in Table 1) server and various workloads generated from three SDSS workloads [40] and 22 standard TPC-H queries [15]. We use another computer to produce database workloads and collect experimental data including query statistics and energy consumption. The energy consumption is measured by power meters (i.e., a 34410A Digital multimeter [41] and a Wattsup power meter [42]). To measure the CPU power, we attach current clamp from the digital multimeter to the CPU power circuit. The energy consumption of the whole server is measured by the Wattsup power meter at a 1Hz frequency. The data server is installed with PostgreSQL 8.3.14 under Redhat 5 (kernel 2.6.9). The DBMS's kernel is modified to provide runtime information such as the estimated energy cost, the data histogram/cardinality and the plan selection.

In the experiments, we set the multi-processing level (MPL) to one (i.e., only one query is in processing at a time in the DBMS). We measure the energy consumption of the entire server and compare it with the estimated energy cost given by the corresponding models. The following metric named *Estimation Error Rate* (EER) is used to quantify the model accuracy:

$$EER = \frac{|E - \hat{E}|}{E} \tag{4}$$

where $\hat{E}$ stands for the estimated energy given by our model and E is the measured energy consumption.

5.2 Experimental Results

We use SDSS and TPC-H benchmarks to simulate queries that retrieve different amounts of data. For example, the SDSS benchmark simulates a large scientific database environment with long queries that touch large amount of data while TPC-H simulates a business data warehouse with a smaller data size. In this way, we could test the energy estimation accuracy using real queries that require processing of various relational operators discussed in our static model.

SDSS Validation: we materialize a database from the published SDSS data. Then we create three workloads: workload I is an equality search based on a sequential table scan; workload II performs a merge join of two tables after range search; and workload III is a range search based on index scan. We repeat each single-query workload for 1,000 times with different search predicates generated randomly. We present the average energy consumption in Figure 10. The difference between the estimated energy and measured energy is very small in almost all cases. The average EERs of the three experiments are 3.32%, 2.66%, and 1.78%, respectively.

TPC-H Validation: To further validate the static model, we conduct a similar experiment for all 22 queries of the TPC-H benchmark. Each workload is a single query extracted from TPC-H tools. The results (Figure 11) show very high accuracy, with an average EER of 7.97%. Query 19 has the highest EER (12.42%) because it has three embedded subqueries in the selection process which leads to system errors

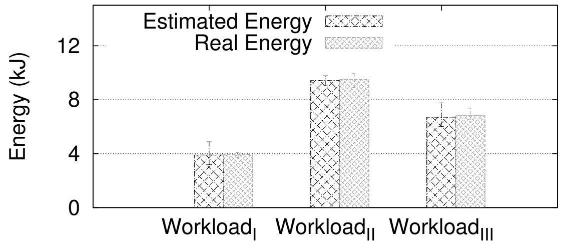

Fig. 10. Energy data of running three SDSS workloads

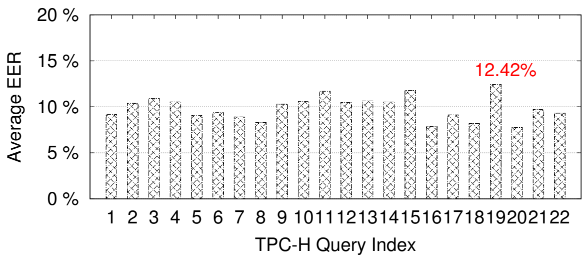

Fig. 11. Estimation errors from the 22 TPC-H queries

that cannot be modeled in Equation 3. Next, we discuss the limitation of the static model.

5.2.1 Limitations of the Models

The above experiments exhibit that the models work effectively in a static setup. However, our modeling task is far from accomplished. When we change some features, such as the MPL of the workload, the model could easily fail. In Figure 12, we change CPU resource availability by introducing CPU-intensive non-database jobs into the system. It causes a serious energy underestimation (blue dash line in Figure 12) and an average EER of 65%. To accurately model energy cost, our model needs to be adaptive to capture the changes of system status.

One might argue that the problem is caused only by the model's failure in capturing the increased baseline power of the system, and can be easily solved by reading a runtime baseline power as a system-level parameter. However, we believe such a solution would still not be robust. First, competition between concurrent queries has profound effects on energy consumption. Second, when such effects are mixed with other situations caused by system state change (such as that in Figure 12), it is almost impossible to tell them apart in the model. To verify our claim in the context of this experiment, we implement and test such an *ad hoc* solution that measures and adds the baseline energy cost to the results of the static models as the estimated energy. The results (pink dash line in Figure 12) clearly show that the *ad hoc* model systematically overestimates the energy with a large error (i.e., average EER reaches 32.67%). The overestimation error comes from resource sharing among concurrent queries.

5.2.2 Source of Errors

Many factors can contribute to errors in energy estimation of a database system. According to our empirical studies, such factors can be separated into three categories.

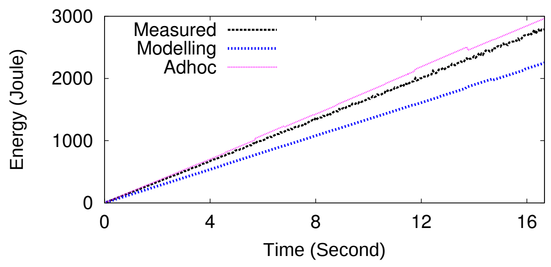

Fig. 12. The performance of the static model under the impact from non-database workloads. It underestimates the energy cost over 65%

System status: runtime state change of the database system and even the OS can cause significant errors in energy estimation. The marginal increase of CPU power is not linear to the CPU utilization. In other words, the parameters (e.g., w_s) should have different values under different system status. At the same time, the resource sharing and the initialization of preemptive non-DBMS jobs can cause the same problem.
Resource demand estimation error: errors are inevitable in estimating the number of tuples (e.g., n in Table 4) for a relational operator in the plan. In our models, such values are provided by the existing query optimizer and are inherently inaccurate. In fact, this is a classic problem in query optimization as the processing time estimation also depends on such quantities [43].
Workload dynamics: a workload generally contains queries with different resource consumption patterns and the interactions among concurrent queries are hard to predict. Workload features may change over time and significantly impact energy cost by changing variables such as the concurrency level and the I/O demand.

To derive an accurate model that minimizes the above errors, we could integrate all relevant factors into an augmented physical model. However, this is an infeasible solution because the model would be too large to meet the fast response requirement and there is no guarantee that the model can locate all the possible factors that have impacts on energy consumption. Instead, we propose to use an online feedback mechanism that adjusts parameters in our static models to minimize those errors.

6 Dynamic Modeling

The main idea of the online estimation approach is: we keep the previous static models, treat the database system as a black-box and take the cost coefficients as variables that reflect the combined effects of all possible system/workload noise sources. We then use a feedback control mechanism to periodically update those parameters using real-time energy measurements. As a result, errors are compensated.

6.1 Online Scheme Design

In this section, we introduce our solution based on the refined Recursive Least Square (RLS) estimator with a directional forgetting factor, to handle estimation errors, following ideas from [8], [44], [45].

In each period with length T_s, we have the operation vector $\vec{o} = \{n_1, n_2, \cdots, n_m, p_1, p_2, \cdots, p_m\}$ to hold values of $2m$ basic operations (i.e., the number of tuples n to-be-computed, the number of pages p accessed by each relational operation) for m operators from each query. Note that, meanings of all notations can be found in Table 2 and the vector $\vec{o}$ is provided by the query optimizer. In general, the RLS scheme builds the coefficient vector $\vec{w} = \{w_1, w_2, \cdots, w_n\}$ to record all to-be-updated parameters and a coefficient $k = \sum_{j=1}^{n} w_j$. In our case, we have $\vec{w} = \{w_s, w_i, \cdots, w_h, w_u, w_r, N_s, N_i, \cdots\}$. Following the routine of constructing the coefficient vector in RLS scheme, let us denote $k_1 = \sum w_x$ and $k_2 = \sum N_x$. The whole coefficient vector for RLS scheme is $\vec{w}' = \{\vec{w}_x, \vec{N}_x, k_1, k_2\}$. $\vec{w}'$ is denoted as $\vec{W}(j)$ for period j. Similarly, we have the whole operation vector $\vec{o}' = \{n_1, n_2, \cdots, n_m, e_1, e_2, \cdots, e_m, 1, 1\}$ (the last two numbers are used to match the operation vector with the coefficient vector) and denote the $\vec{o}'$ at period j as $\vec{O}(j)$. In each period, the energy consumption of the server (E) is measured. The model generates the baseline energy as $e(j)$ from the measurement of the last $j-1$ periods and the measured energy E as follows:

$$e(j) = \frac{((j-1)e(j-1) + E)}{j} \tag{5}$$

we set the initial energy consumption as $e(0) = 0$. The next step is to use this estimator to find the values of energy cost parameters. The coefficient vector $\vec{W}(j)$ is updated as follows:

$$\vec{W}(j) = \vec{W}(j-1) + \frac{\epsilon(j)\vec{O}^T(j)\mathbf{M}(j-1)}{\lambda + \vec{O}(j)\mathbf{M}(j-1)\vec{O}^T(j)} \tag{6}$$

where $\epsilon(j) = e(j) - \vec{O}^T(j)\vec{W}(j)$ is the estimation error. $\vec{O}^T(j)$ is the transpose of the vector $\vec{O}(j)$, $\mathbf{M}(j-1)$ is the covariance matrix of vector $\vec{O}(j)$, and $\lambda \in [0,1]$ is the forgetting factor – a smaller λ enables the estimator to forget the history faster. The RLS estimator adapts itself so that $\epsilon(j)$ is minimized. When the two variables, $\vec{O}$ and $e(j)$, are jointly stationary, this algorithm converges to a set of tap-weights which, on average, are equal to the Wiener-Hopf solution [46]. The following routines are repeated at the beginning of every period j:

i. Collecting data, including the operation vector $\vec{O}(j)$ and previous baseline energy $e(j-1)$;
ii. Computing $\vec{W}(j)$ based on the data and Equation (6).

The RLS estimator needs around 12 microsecond on average for computing the parameter, as measured in our experiments. Compared to the average query processing time (around 20 seconds), the computational overhead is very small. It is also robust despite system dynamics and workload variations. Based on the results obtained from the static models, the initial values for the energy parameters ($\vec{W}$) in our testbed are $\vec{W}(0) = \{0.00768, \cdots, 1153, 2109, 2654\}$, as shown in Table 3.

The length of T_s implicitly affects the accuracy of the RLS model. It is relevant to the query arrival rate. If the query arrival rate is high, T_s needs to be set smaller to sample sufficient variance. Otherwise, we could make it longer to avoid excessive computational overhead. In our experiment, we set it to be $1/9$ seconds, the same sampling frequency as the energy cost measurement. In all, we build a two-layer estimation model called RLS model.

6.2 Experimental Setup

We use the same test environment in Section 5.1. However, as the enhanced online model is meant to provide a high estimation accuracy despite the workload/system noises, we design synthetic tests that simulate such noises. In all such experiments, we set the MPL $\in [2, 31853]$ (31853 is the default max threads allowed in Linux) to create a more realistic database runtime environment, in which multiple queries are processed concurrently. We simulate the impact of different error sources described in Section 5.2.2 using the following three test cases.

Type I: to test the accuracy of RLS model under workload and system noises, we define this type of workload with different data sharing patterns among concurrent queries. Specifically, we have the *share-everything* (SE) and *share-nothing* (SN) workloads. The SE workload consists of queries with small computation and a considerable amount of data shared with other queries. The SN workload consists of queries with long processing time and little data shared with other queries. The results are in Figure 13.

Type II: poor estimation of data distribution (e.g., data histograms) in database tables is the main reason of the estimation error in resource demand [43]. We design this test to examine the performance of our model under good and poor resource estimation from query optimizer. This test contains: (i) *deterministic access* (DA) workload that always searches similar data from the table; and (ii) *random access* (RA) workload that randomly touches all spectra of the data domain. Upon running the DA workload, the query optimizer can learn the data distribution quickly due to the same data location in the table, thus has more accurate resource estimation for the rest queries. For the RA queries, query optimizer needs to collect much more information using a significantly longer time, which leads to inaccurate resource estimation. The results are in Figure 14.

Type III: in a real-world database server, other applications running concurrently with database processes (e.g., httpd and php processes) may affect the resource availability. We design this test to simulate those noises in two scenarios. The first one is to test the estimation performance while competing for the CPU resource. We run multi-thread Fibonacci programs (*Fibo*) concurrently with the Postgres application. The second one is to change I/O bandwidth at runtime using R/W programs (*R/W*). R/W is an I/O-intensive program that frequently reads/writes large files, thus competing with the DBMS for I/O bandwidth. The results are in Figure 15 and 16.

We compare the performance of the RLS model with two baselines: the static model in Section 4 and the *ad hoc* model mentioned in Section 5.2.1. Note that, since we have multiple queries running in the system now, the average EER that we used to evaluate the system's performance is redefined as the average of the EERs of all involved queries.

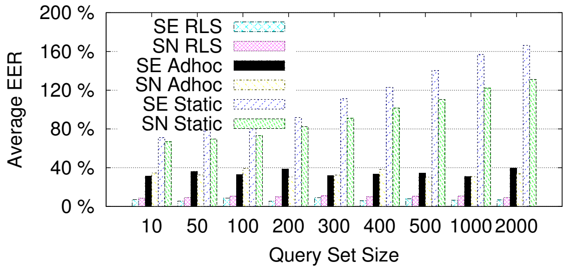

Fig. 13. Estimation accuracy of three models in nine workloads with different data sharing patterns using the SN/SE workload for case study Type I

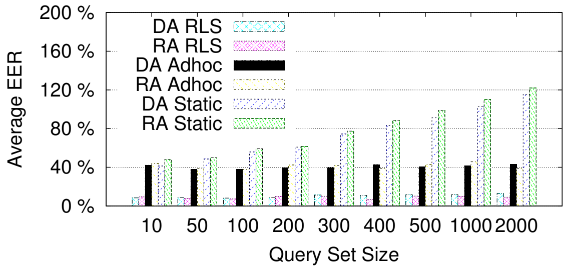

Fig. 14. Estimation accuracy of three models in nine workloads with different data access patterns using the DA/RA workload for case study Type II

6.3 Experimental Results

Results of Type I. In such experiments, we create nine workloads with different sizes to test three models – RLS model, *ad hoc* model and static model. As shown in Figure 13, the EERs generated by the RLS model are significantly smaller than that of the static model for both SE and SN workloads, with an all-round average EER of 8.89% and 6.93%, respectively. Clearly, RLS model can effectively handle the correlation among queries. The *ad hoc* solution, which could partially capture the interactions among queries, shows a better performance than the static model. However, it is not compatible to the full-fledged RLS model – its EER often triples (the average EERs are 33.54% and 34.78% for the two workload types). Another observation here is, the SN workload usually causes more errors than the SE workload in RLS model but less errors in static model. Our explanation is that the SN workload provides more fluctuations in the energy consumption than the SE workload does during the execution. Thus, depending on the value of forgetting factor λ, the parameter estimator generates more errors than that in SE workloads. Static model favors SN workload since it has no knowledge of resource sharing among queries and the estimation in the SN workload only needs to add up the energy consumption of each individual query.

Results of Type II. As shown in Figure 14, the RLS model beats other two models in accuracy when handling DA

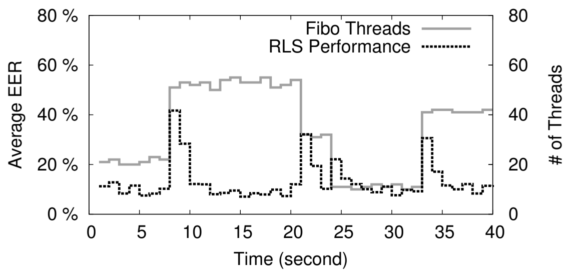

Fig. 15. Model performance in case study Type III using the workload Fibo

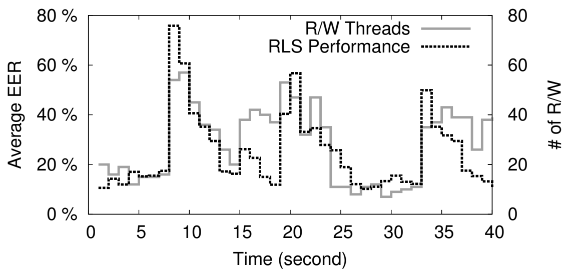

Fig. 16. Model performance in case study Type III using the workload R/W

workload – the average EER is 7.13% with the highest being 11.9%. For the DA workload, queries always visit the same part of the table therefore it leads to very high cache hit rate. That is likely to be the reason why the estimation errors of static model show (roughly) a linear increase in Figure 13. For the RA workload, although the query optimizer could produce large error in resource estimation, our dynamic model can capture the trend of such errors and compensate them. The EERs are lower than 10% for most cases – the average EER is 7.26% with the highest EER being 11.07%. For both workloads, the performance of RLS model is good as it is designed to capture the variations from resource estimation.

Results of Type III. In Figure 15a, the system starts with 20 Fibo processes. This number rises to 54 at the seventh second, drops at the 24th second to 10, and increases to 40 at the 33rd second. The change of CPU utilization leads to a high resource estimation error. By comparing the number of running Fibo processes (black line) and the energy estimation error (grey line), the RLS model can capture the trend of such change and react within a short period of time (i.e., less than three seconds). It keeps its average performance by keeping a 90% accuracy of the energy cost estimation.

When it comes to the I/O resource competition, the measured energy consumption unpredictably increases with the number of I/O-intensive applications (RW) in Figure 16. The reason is that the performance bottleneck of data processing is still the I/O bandwidth. When this critical resource is “stolen”, most queries are in halt to wait for I/O resource, which results in huge wasted energy consumption. As a result, the estimation performance is greatly affected by those applications that compete with DBMS for the I/O resource, such the data change at the 7th and 15th second in Figure 16. When the system status change tends to be steady, such as in time period of 15 – 19 and 34 – 40, RLS model tries to get its performance back within a few periods (a little bit longer than that in Fibo). In all, RLS model could handle the noise from I/O bandwidth change to some degree.

Comparison with other estimation methods. To highlight the benefit of DBMS-level energy models, we compare them with other energy estimation models deployed at the OS-level. *System* is a system level energy estimation model

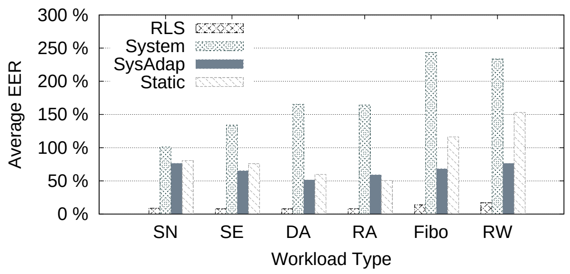

Fig. 17. Comparison of static and RLS models with other energy estimation models

[20], and *SysAdap* is an enhanced *ad hoc* OS-level energy estimation model [23]. We use those models to estimate energy consumption of workloads used in above cases. The results are shown in Figure 17. It is not a surprise that *System* has the worst estimation performance since it considers the energy cost of all hardware operations which may not be caused by DBMS operations. Comparing with *System*, *SysAdap* shows a relatively better performance in those cases because it periodically corrects itself according to the detected DBMS energy cost. However, the performance of *SysAdap* is only compatible with the performance of the *Static* model. The RLS model gives highly accurate estimation in all cases. Thus, for energy cost estimation, we need to build the estimation model inside the DBMS.

Black box validation. We also validate our model within a closed benchmark environment generated from TPCC-UVa[3], as mentioned before. Since we cannot change data distribution or the query composition of the workload, this serves as a perfect tool for black box testing. As seen in Figure 18, the average EER of our dynamic model is around 10% in a 100 GB DBMS configuration compared to the static model’s EER of 51.4% and the ad hoc model’s 23.62%. This clearly shows that our RLS model is robust even under a random workload.

Effects of forgetting factor λ. Another interesting experiment is the what-if analysis on the important forget factor λ in our dynamic model. The results of this experiment could help us fully explore the effectiveness of the model and make recommendations on the choice value of λ. For that purpose, we create six workloads – two from SDSS queries

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

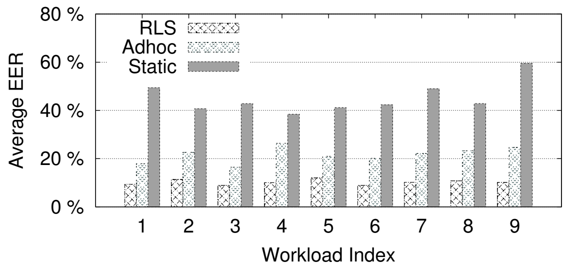

Fig. 18. The average EER of running multiple TPC-C workloads in TPCC-Uva

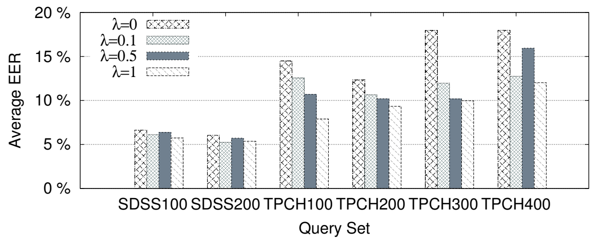

Fig. 19. Accuracy of the RLS model under six different workload sets and four different λ values

and four with various combinations of TPC-H queries – to provide a diversified testing environment. As seen in Figure 19, a smaller forgetting factor has more negative effects on the accuracy of the RLS model. For the SDSS workloads (the first two histogram clusters in Figure 19), the results are stable without showing much difference under different λ. Our explanation is: most queries in SDSS are I/O-bound due to the sheer size of the database table - this creates a very static situation in terms of the energy consumption. However, in a dynamic environment, if the system states follow a historical trend (e.g., those of coarse-grained parallel workload or a deterministic access workload), increasing the value of λ gains significant benefits in terms of model accuracy. Thus, we suggest the maximum value (i.e., 1.0) of λ be used in order to obtain more historical data and higher accuracy of the model.

7 Impact of Energy-aware Query Optimization

Our previous work [12] proposed a composite query cost model that considers both latency and energy of the query plans. Furthermore, the model can be configured to reflect the energy/performance tradeoff the DB Administer wishes to adopt. Obviously, the effectiveness of such a cost model depends on how accurately we estimate the energy/latency of the visited plans. In this section, we explore the impact of our energy models on query optimization towards energy consumption. First, we provide an energy efficiency study based on the energy consumption data of executing queries optimized by three different query optimizers. Workloads are processed by a PostgreSQL system with three different flavors of query cost model in its query optimizer – *Traditional* (the original PostgreSQL cost model that only considers query latency), *Power-aware* (the optimizer

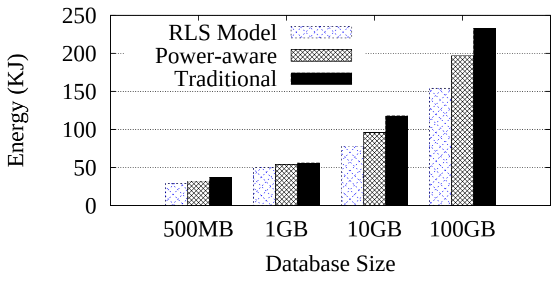

Fig. 21. The energy consumption of different query plans with different optimization goals

with a query cost model towards maximal power efficiency but with a simple energy estimation scheme), and *RLS model* (same optimizer as in 'Power-aware' but the dynamic energy estimation model proposed in this paper). To further explore what exactly changed in the plan selection, we first reveal the detailed processing paths from those three optimizers for one query (Q10 from the standard TPC-H benchmark). We then present results of other affected queries, and finally discuss the rationale behind the results.

7.1 Enhanced Energy Savings

We first present the energy consumption results from the three aforementioned query optimizers. In this experiment, we run the same workload in databases with different sizes, namely 500MB, 1GB, 10GB, and 100GB. Due to the execution plans selected by the optimizers towards different optimization goals, the energy consumption performance varies, as shown in Figure 21. The energy-aware optimizer picked query plans with the least energy consumption. The energy savings are in the range of 14.7% to 33.3%, depending on the database size. This, on the other hand, comes with the cost of a performance loss ranging from 8.3% to 13.5%. Without an accurate energy estimation scheme, the *Power-aware* optimizer stands in between *Traditional* and *RLS* model for both energy consumption and performance. Note that, in some cases, the highest performance (i.e., latency) query plan can also be the most power efficient and energy efficient plan. In that case, all three optimizers would achieve the same energy saving. However, at the entire workload level, the accurate energy cost estimation helps reveal the tradeoffs in the plan selection to find more energy-efficient plans. Thus, we advocate integrating energy cost estimation model into the current optimizer to analyze the energy cost profile of different configurations in DBMS.

7.2 Plan Selection

To further explore the impacts of our model on query optimization, we also study the patterns of plan selection. For example, the chosen plans of Query 10 of the standard TPC-H benchmark are different in different optimizers (Figure 20). In particular, when the optimization goal changes from performance (Figure 20(a)) to power (Figure 20(b)), we see indexed scans replaced by sequential scans, and hash joins by sort merge and nested loop joins. With more

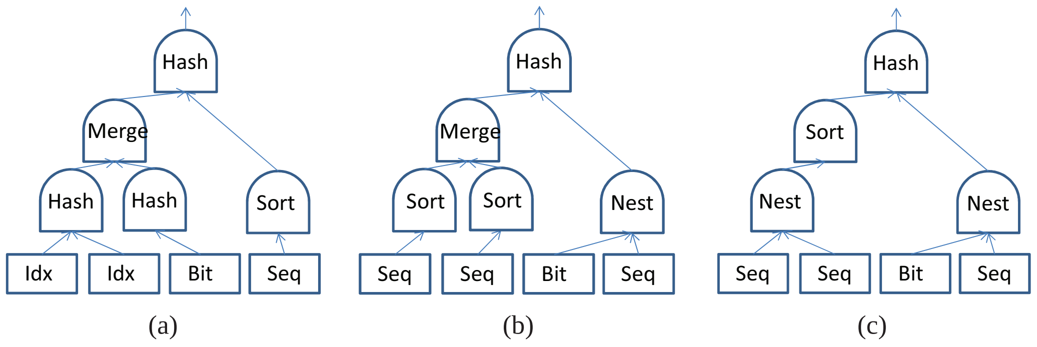

Fig. 20. Three selected paths from three optimizers (i.e., (a) Traditional, (b) Power-aware [12] and (c) RLS model) using TPC-H Q10 under the 100GB database size

TABLE 5
Change of plan selection from Traditional to the RLS model under different database sizes.

	Scan				Join			
Size	Sequential	Index	Bitmap	Unchanged	Nestedloop	Sort-merge	Hash	Unchanged
0-4KB	48.7%	3.6%	0.2%	47.5%	34%	0%	0.1%	65.9%
4KB-100MB	39.2%	26.6%	6.7%	27.5%	16.4%	15.5%	2.1%	76%
100MB-1GB	9.4%	49.1%	12.2%	29.3%	16%	17.3%	10%	56.7%
1GB-1TB	4.7%	42.6%	17.3%	46%	8.3%	34.6%	9.2%	47.9%
1TB-10TB	0%	21.7%	3.7%	74.6%	0.3%	56.7%	14.4%	28.6%

accurate energy estimation (Figure 20(c)), the two sort merge joins are further replaced by nested loop joins. This confirms our findings in Section 4.2 that hash join consumes more energy and sequential scan sometimes has better energy efficiency than indexed scan. Similar observations are made in many other queries.

We present all the plan change statistics in Table 5 using all 5000 queries from Section 6.3. Each number in Table 5 represents the percentage of selected operator after changing the optimization goal from performance (*Traditional*) to energy efficiency (*RLS* model). For example, under database size of 1GB-1TB, when we change the optimizer from *Traditional* to *RLS* model, the same algorithm was picked for 47.9% of the join queries. Meanwhile, we also see 34.6% of the queries changed to sort merge join (from other join algorithms). We have the following observations from Table 5. Although sometimes the energy-aware optimizer chooses the same algorithm as *Traditional*, on average around 54% of the cases we saw a change of selected plan. This clearly shows the great opportunities of reaching different energy/performance tradeoffs by extending the search space into the energy dimension. When the size of the target table is small, more sequential scans are chosen, and as table size increases, index scan is preferred. This exactly matches the pattern shown in Figure 3. When the data table is sufficiently large, the marginal energy cost of visiting hash table is relatively small. Thus, it is more likely to be chosen in large table joins than small table joins. Sort-merge shows similar trends: in over 56.28% of the join operators, sort-merge joins are chosen instead to reduce the energy consumption due the much less CPU computation and comparable similar I/O references, compared with the hash join. The nested-loop join is only selected when the reference data table is small enough to sit in the database buffer. In this case, the nested-loop has the minimum CPU operations and acceptable I/O references.

8 Conclusion

This paper argues for the importance of building accurate and robust models for energy cost estimation in database systems. For that purpose, we conducted system identification experiments on a database server to explore the essential components of possible energy models. Based on those findings, we proposed and evaluated a two-level energy estimation model: we started from a series of heuristic models that describe the unit-energy cost of individual relational operators and important database operations following our empirical results, and used a RLS estimator to tune cost coefficients of static models according to the feedback signal. Such a two-level modeling solution can tolerate high system dynamics and workload variations. In summary, our models are found to be effective as the average estimation error is below 10%. We strongly believe our work serves as the basis for energy-aware query optimization – a key component in building energy-efficient database management systems.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments. This research was supported by the US National Science Foundation (NSF) grants IIS-1117699, IIS-1156435, and CSR-1143607.

References

[1] F. Ahmad and T. N. Vijaykumar, "Joint optimization of idle and cooling power in data centers while maintaining response time," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 243–256, 2010.
[2] P. Bohrer, E. N. Elnozahy, and et al., "Power aware computing," R. Graybill and R. Melhem, Eds., 2002, ch. The case for power management in web servers, pp. 261–289.

[3] L. Benini, R. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on VLSI Systems*, vol. 8, pp. 299–316, 2000.

[4] A. Berl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. de Meer, M. Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *Comput. J.*, vol. 53, no. 7, pp. 1045–1051, 2010.

[5] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. of the MICRO*, 2003, pp. 93–105.

[6] B. Chandramouli, W. C. Hsieh, J. B. Carter, and S. A. McKee, "A cost model for integrated restructuring optimizations," *J. Instruction-Level Parallelism*, vol. 5, 2003.

[7] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *SoCC*, 2010, pp. 39–50.

[8] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," in *IEEE Real-Time Systems Symposium*, 2008, pp. 303–312.

[9] 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*, vol. 1, no. 2, pp. 1229–1240, 2008.

[10] R. Agrawal, A. Ailamaki, and et al., "The Claremont Report on Database Research," *Communications of ACM*, vol. 52, pp. 56–65, Jun. 2009.

[11] W. Lang, R. Kandhan, and J. M. Patel, "Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race," *IEEE Data Engineering Bulletin*, vol. 34, no. 1, pp. 12–23, 2011.

[12] Z. Xu, Y. Tu, and X. Wang, "Exploring power- performance tradeoffs in database systems," in *Proc. of ICDE*, 2010.

[13] M. Poess, R. O. Nambiar, K. Vaid, J. M. Stephens, K. Huppler, and E. Haines, "Energy benchmarks: a detailed analysis," in *e-Energy*, 2010, pp. 131–140.

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

[15] Transaction Processing Performance Council, http://www.tpc.org.

[16] M. Kunjir, P. Birwa, and J. Haritsa, "Peak Power Plays in Database Engines," in *Proc. of the EDBT*, 2012.

[17] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 103–116, Oct. 2001.

[18] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: managing energy as a first class operating system resource," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. 5, pp. 123–132, Oct. 2002.

[19] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *Computer*, vol. 37, no. 11, pp. 68–74, Nov. 2004.

[20] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proceedings of PPoPP'05*. New York, NY, USA: ACM, 2005, pp. 186–195.

[21] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proc. of the ISCA*. New York, NY, USA: ACM, 2011, pp. 319–330.

[22] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the USENIX'09*, 2009, pp. 28–28.

[23] X. Wang, K. Ma, and Y. Wang, "Adaptive power control with online model estimation for chip multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1681–1696, Oct. 2011.

[24] J. B. Carter and K. Rajamani, "Designing energy-efficient servers and data centers," *IEEE Computer*, vol. 43, no. 7, pp. 76–78, 2010.

[25] Z. Abbasi, G. Varsamopoulos, and S. K. S. Gupta, "Tacoma: Server and workload management in internet data centers considering cooling-computing power trade-off and energy proportionality," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 2, pp. 11:1–11:37, Jun. 2012.

[26] R. Alonso and S. Ganguly, "Energy Efficient Query Optimization," Matsushita Info Tech Lab, Tech. Rep., 1992.

[27] G. Graefe, "Database servers tailored to improve energy efficiency," in *Proc. of EDBT Workshop*, ser. SETMDM '08, 2008, pp. 24–28.

[28] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan, "Energy efficiency: The new holy grail of data management systems research," in *CIDR*, 2009.

[29] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *Proc. of the international conf. on management of data*, ser. SIGMOD '10, 2010, pp. 231–242.

[30] M. Poess and R. O. Nambiar, "Tuning servers, storage and database for energy efficient data warehouses," in *ICDE*, 2010.

[31] ——, "Power Based Performance and Capacity Estimation Models for Enterprise Information Systems," *IEEE Data Engineering Bulletin*, vol. 34, no. 1, pp. 34–49, 2011.

[32] Z. Xu, Y. Tu, and X. Wang, "Dynamic energy estimation of query plans in database systems," in *Proc. of ICDCS*, 2013.

[33] M. M. Astrahan, H. W. Blasgen, and et al., "System r: Relational approach to database management," *ACM Transactions on Database Systems*, vol. 1, pp. 97–137, 1976.

[34] S. Christodoulakis, "Implications of certain assumptions in database performance evaluation," *ACM Trans. Database Syst.*, vol. 9, no. 2, pp. 163–186, 1984.

[35] L. F. Mackert and G. M. Lohman, "R* optimizer validation and performance evaluation for distributed queries," in *VLDB*, 1986, pp. 149–159.

[36] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis, "Towards energy-efficient database cluster design," *PVLDB*, vol. 5, no. 11, pp. 1684–1695, 2012.

[37] P. Mahadevan, P. Sharma, and et al., "A power benchmarking framework for network devices," in *Proc. of the NETWORKING*, 2009, pp. 795–808.

[38] Z. Xu, "Power cost estimation in dbms, a comprehensive study," Department of Electrical and Computer Engineering, The Ohio State University, Tech. Rep. OSU-ECE-12-002, June 2012.

[39] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: an os facility for fine-grained power and energy management on multicore servers," in *Proc. of ASPLOS*, 2013, pp. 65–76.

[40] Sloan Digital Sky Survey, http://cas.sdss.org/dr7/en/.

[41] "34410a digital multimeter," http://goo.gl/0GUX7.

[42] "Watts up power meter," http://goo.gl/AI7so.

[43] S. Chaudhuri, "An overview of query optimization in relational systems," in *PODS*, 1998, pp. 34–43.

[44] A. Vahidi, A. Stefanopoulou, and H. Peng, "Recursive least squares with forgetting for online estimation of vehicle mass and road grade: theory and experiments," *Vehicle System Dynamics*, vol. 43, no. 1, pp. 31–55(25), 2005.

[45] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," *SIGARCH Comput.*, vol. 37, pp. 314–324, June 2009.

[46] J. B. Lawrie, "A brief historical perspective of the Wiener–Hopf technique," 2007.

Zichen Xu received his BE degree in Computer Technology from Beijing University of Posts and Telecommunications, China (2007), and his MS degree in Computer Science from University of South Florida (2009). He is currently a PhD candidate in the department of Electrical and Computer Engineering at the Ohio State University. His research spans in energy efficient data processing, energy/performance modeling, and power-performance balanced computing system.

Yi-Cheng Tu received a Bachelor's degree in horticulture from Beijing Agricultural University, China, and MS and PhD degrees in computer science from Purdue University (2003; 2007). He is currently an associate professor in the department of Computer Science & Engineering at the University of South Florida. His research is in energy-efficient database systems, scientific data management, high performance computing and data stream management systems. He received a CAREER award from US National Science Foundation (NSF) in 2013.

Xiaorui Wang received the D.Sc. degree in computer science from Washington University in St. Louis in 2006. He is currently an associate professor in the Department of Electrical and Computer Engineering at the Ohio State University. He is the recipient of the US Office of Naval Research (ONR) Young Investigator (YIP) Award in 2011 and the US National Science Foundation (NSF) CAREER Award in 2009. His research interests include computer architecture and systems. He is a member of the IEEE and the IEEE Computer Society.