Power Modeling in Database Management Systems

Zichen Xu [1], *Yicheng Tu* [2], *Yefu Wang* [3], *Xiaorui Wang* [1]
[1] *Electrical and Computer Engineering, The Ohio State University, U.S.A.*
{xuz,xwang}@ece.osu.edu,
[2] *Computer Science and Engineering, University of South Florida, U.S.A.*
ytu@cse.usf.edu,
[3] *Siemens, U.S.A.*
ywang38@eecs.utk.edu

USF CSE Technical Report
Number 11-055
November 2011

Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
URL: http://www.cse.usf.edu/

Power Modeling in Database Management Systems

Zichen Xu [1], Yicheng Tu [2], Yefu Wang [3], Xiaorui Wang [1]
[1] Electrical and Computer Engineering, The Ohio State University, U.S.A.
{xuz,xwang}@ece.osu.edu,
[2] Computer Science and Engineering, University of South Florida, U.S.A.
ytu@cse.usf.edu,
[3] Siemens, U.S.A.
ywang38@eecs.utk.edu

University of South Florida, CSE, CSE/11-055
November 2011

Abstract

Data centers consume large amounts of energy. Since databases form one of the main energy sinks in a typical 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 designing such systems. In this paper, we report our recent efforts on this topic, with a focus on the power estimation of query plans during query optimization. We start from a series of physical models designed for evaluating power cost of individual relational operators. Such models form the basis for a composite power model for the query plans. Key parameters for the query power model are generated by using linear regression tools upon running experiments with simple workloads in a static environment. Our solution can effectively tune the model parameters at runtime to reach high accuracy of power estimations. The models are implemented in the PostgreSQL kernel and evaluated with a comprehensive set of experimental workloads generated from multiple TPC and scientific database benchmarks.

1 Introduction

Data centers, which host a lot of the world's computing resources, have been lambasted as the "SUVs of the tech world" for their enormous consumption of energy. In 2006, data centers in the U.S. alone consumed roughly 61 billion kilowatt hours (kWh) of energy, which is about 1.5% of all electricity consumed by the country [17]. Meanwhile, the energy demand of data centers grows at a rate of 12% per year, leading to approximately 100 billion kWh of energy consumed in year 2011 - this translates into an annual electricity cost of $7.4 billion [12]. Therefore, energy consumption becomes one of the main optimization goals in data center design and maintenance.

Recently, energy conservation has attracted much attention from the database community, as database management systems (DBMSs) and their applications are found to be the main consumers of energy in a typical data center [24]. Work in this area ranges from single node power optimization, which combines time performance and energy usage as the target in query optimization, to power management policies in distributed databases [31, 27, 5, 25].

A common theme in such research is to design database systems with energy consumption as a first-class performance goal. Therefore, the quantification of the energy cost of a database becomes an important task in designing such systems. We believe energy cost estimation carries high technical significance and can be done at two levels. At the single query level, the energy cost of alternative query execution plans is indispensable in making decisions related to query optimization [31] and query rescheduling [19] in energy-aware DBMSs. In an energy-aware query optimizer advocated in our previous work [31], a query plan is explicitly evaluated by both its (estimated) energy consumption and performance. In such a scheme, accurate energy estimation is critical in ensuring the effectiveness of query optimization. At the database system or even the data center level, estimating the energy consumption of a workload is an essential part of runtime resource management toward energy conservation, especially for systems running on virtual resources [14]. In general, power/energy estimation is known to be a critical component in energy-aware computing systems. According to [13], all dynamic power management (DPM) policies can be divided into two groups:

predictive schemes and stochastic schemes. For both cases, mechanisms to quantify system energy consumption are needed. In this paper, we report results of our explorations in *energy cost estimation in databases, with a focus on the power modeling of query plans within the DBMS*. A salient feature of our power model is that it works under system/environmental dynamics by automatically adjusting its parameters. We believe that such work is required for energy-aware query optimization, and it will also build the foundation for predicting power consumption of workloads on the system level.

Specifically, our work focuses on *power* cost instead of *energy* cost for two reasons. First, power itself has been regarded as a primary optimization goal in data centers. Low power generally translates into low energy consumption. For example, we have found that in databases there are many query plans that require much less power with the cost of little performance degradation [31]. Furthermore, energy consumption of cooling systems also decreases when system runs in low-power modes. A recent study [2] indicates that for every Watt saved for powering the servers, at least another half a Watt can be saved from the cooling systems. Second, energy cost of a task can be easily obtained from its power cost and running time (i.e., energy = power $\times$ time). By focusing on power, we can, to some extent, separate the errors inherited from time estimation from those generated by our power estimation models. Note that running time estimation is a classic problem in query optimization. Existing solutions to that problem can be integrated with our work on power modeling to estimate energy consumption of database systems.

In this paper, we first introduce a physical model to estimate the power consumption of queries running in a database server. We build such a model from low-level models that describe the power consumption of relational operators via studying the (hardware) resource consumption patterns of operator processing algorithms. Parameters of such models are derived from a training query workload using classic regression tools. Such models show high accuracy in predicting query power consumption when dealing with simple workloads running under a stable computing environment. However, the value of the key parameters (e.g., number of Watts to process an indexed tuple) of the model is dependent on system information (e.g., number of CPU cores currently in use) and workload statistics (e.g., table cardinality, query arrival rate, etc.). Thus, in a high intensity workload testing environment, the model is no longer feasible because it cannot handle the unpredicted multi-variables in the dynamic environment. In this case, we need to deploy some dynamic strategies in the implementation but it is beyond the scope of this paper.

We implemented and evaluated our power models in the PostgreSQL system. For that purpose, we developed a workload engine that produces training/testing workloads and data sets derived from various TPC and scientific database benchmarks. We validated our models using a comprehensive set of workloads generated from that engine by comparing power predictions (provided by our model) with real power measurements (using power meters). Our results show that our model estimates power consumption of query plans with high accuracy - estimation error is less than 10% in almost all the test cases. As an exploratory study of this important topic of query energy quantification in DBMSs, our work is focused on power estimation for query plan evaluation and the results are tested in a relatively small database server. Despite such limitations, we believe this work provides a general methodology for database power estimation and can be applied to a wide range of database system environments.

2 Related Work

The steep increase of energy consumption of computers and tighten budget on price/unit computing have made power management a critical issue in the system design and implementation. Inspired by this issue, lots of researchers have created tremendous work as [8, 32, 21, 7] focused on hardware and computing platforms. While on the other hand, much interest has been put into power-aware software applications for real-time system or on mobile platforms, such as [6, 10] in the past few years. Database system, being one of the largest consumers of computing resources and energy in modern data centers [24], is the targeted object that needs to be modified to be energy-efficient.

Energy consumption in DBMSs is a relatively new topic in the database community. Two papers published in CIDR'09 are closely related to this proposal: [12] presents a wide range of high-level ideas on how energy efficiency can be improved in databases. While many of those ideas are similar to ours in philosophy, this paper does not provide any technical details on the challenges and solutions in realizing those ideas. Supported by initial experimental results, [19] presents two specific techniques to save energy in databases: controlling CPU frequency and re-scheduling user queries. The results presented in [27] shows that high energy efficiency coincides with high performance in databases. Raghunath and Meikel, though mainly focus on power efficiency on storage system of transaction system, discussed many ideas on storage configuration and experiments in order to improve the DBMS's power performance [24]. [18] gives an overview of energy efficiency query processing, which provides an estimation metrics similar as our general

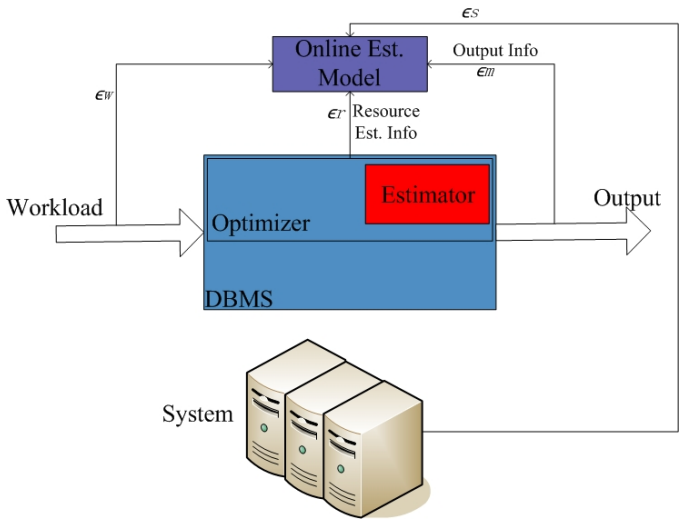

Figure 1: The experimental platform with possible error sources

power consumption Eq. 2.

Relational operations' cost modeling is a conventional problem. The discussion about those topics can be traced back to late 70s. Astrahan and etc. discussed the initial idea in system R[3]. Stavros Christodoulakis summarized the early work and provided the basic estimation assumptions in 1984 [11], Lothar and Guy extended the work into distributed environment later on [22]. Standing on their shoulders, we follow the same mechanism to build up our cost model based on similar assumption. However, since we are modeling a different object, the variants and target objectives are no longer the same.

Dynamic Power Management (DPM) is an overall concept to build up power optimal model in the system level. Many techniques can be found in [4]. [15] presents randomized online algorithms for spin-block and snoopy-caching problems, which we have considered to apply to the dynamic system environment problem here. [23] introduces a finite-state, abstract system model for power-managed systems using Markov decision processes. Within this model, the problem of finding policies becomes stochastic optimization problem of finding optimal tradeoff between performance and power. However, the model's preprocessing time is so large that we cannot endure this penalty in the gain of limited accuracy. Those work inspires us in creating our RLS model in the predictive scheme and refine it in our environment.

3 System Implementation

3.1 Hardware

Our testbed for model validation consists of two servers for different purposes. The one called "worker" contains a 3.0 GHz quad-core CPU Xeon X5365, 4GB of 667 MHz DDR3 memory, and a 2TB 7200RPM hard drive, as shown in Table 2. It is used to run the database server and thus the target for power modeling. The other one called "monitor" is responsible for runtime collection of experimental data including query statistics and power consumption. Power measurement is done by power meters (i.e., USB Oscilloscope DSO-8502 and Watts' Up power meter) attached to "worker" and linked to the "monitor" via USB connections for data reading. Specifically, the DSO-8502 is used to measure CPU power and the Watts' Up for that of the entire database server. The structure of the "worker" is sketched in Fig. 1.

3.2 Software

Our "worker" machine is installed with PostgreSQL 8.3.14 as the DBMS under Ubuntu 9.10. The DBMS's kernel was hacked in order to provide us detailed runtime system information such as estimated cost, data histogram and plan

selection. Along with the DBMS, we also implemented in the "worker" a workload generator based on TPC-H, TPC-C benchmarks[1] and datasets from the SDSS database [2] – a well-known large-scale scientific database. The workload generator creates a comprehensive set of workloads designed to simulate the effects of major factors that impact power modeling, and are used to test the effectiveness of our models.

3.3 Experimental Setup

We conducted a series of experiments to verify the models listed in Table 3. As the first step of the modeling process, we created an ideal environment to run experiments such that we can concentrate on verifying the model structures and obtaining initial values for the model parameters from the a set of training workloads. In the setup, we followed the assumptions to create such an environment in which our estimation of the number of operations.

We used a dedicated server for our experiments and fed the database with simple workloads that consist of very few types of queries. In such experiments, we set the multi-processing level (MPL) to be one, i.e., only one query is processed at a time. We measured the real power of the entire server and compared it with the estimated power given by the corresponding models. A metric named *Estimation Error Rate* (EER) is used to quantify the model accuracy. Specifically, EER is defined as

$$EER = \frac{|C' - C|}{C} \tag{1}$$

where C stands for the estimated power cost and C' is the actual power measured by a power meter.

4 System Identification

In a traditional DBMS [22], 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 required for executing the plan and the resource consumption in each such operation. The relevant numbers of basic operations include: number of pages read or written to disk (N_{pages}), number of tuples (N_{tuples}) and/or indexed tuples ($N_{indexed\_tuples}$) to process in the CPU, and the number of bytes transmitted via networking system (N_{msg}). Such a model can be a starting point for power estimation in DBMSs. Specifically, as mentioned in a recent position paper [18], the power cost of a plan can be expressed as

$$P = W_{cpu} \times \frac{N_{tuples}}{T} + W_{cpu\_index} \times \frac{N_{indexed\_tuples}}{T} + W_{I/O} \times \frac{N_{pages}}{T} + W_{msg} \times \frac{N_{msg}}{T} \tag{2}$$

where quantity T is the query processing time, and $W_{...}$ are tunable system parameters. Among them, W_{cpu} and W_{cpu_index} are related to the energy consumed by CPU, $W_{I/O}$ by the storage system, and W_{msg} by the networking system. In our paper, we only focus on one single server scenario and therefore the last item with communication cost can be ignored.

As a general model, Eq. (2) needs to be refined for accurate power estimation by considering the following two aspects.

4.1 Observations on Hardware

First, there is a need to elaborate on the roles of different hardware components in a typical database server play in power consumption. In our experiments, we measured the power consumption of major hardware components in a database server when the system runs in an idle state (when no query is being processed) and when it is fully utilized (Table 1). It is easy to see that CPU contributes most to the active power used by the system (about 99%), and the difference between its peak and idle power is large (147.15-88.2 = 59 watts). All other components consume almost the same power no matter how intensive the workload is. Such results also verify the findings reported in other work on database energy use [19, 27]. To further reveal the power patterns of system components, we also recorded their power consumption under different workload intensities. Fig. 2 shows the results: the power consumption of CPU increases monotonically with the increase of the utilization while very little difference can be observed for the storage system. For different workloads to execute sequantial read and random read in the storage system, as we could observe

[1] http://www.tpc.org/

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

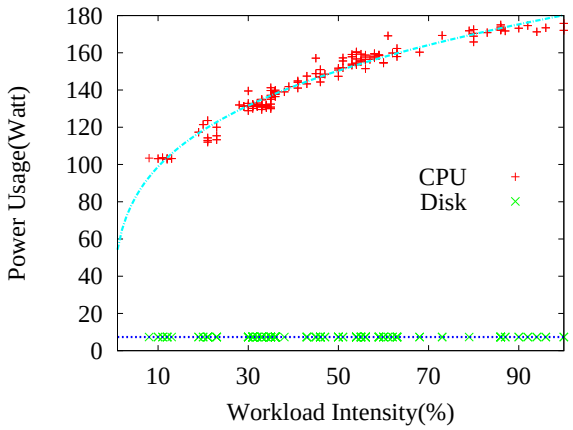

Figure 2: CPU and hard disk power consumption under different levels of workload intensity

Table 1: Power consumption (in Watts) of major hardware components in our database server.

Component	Peak Power	Idle Power
CPU: Xeon X5365	147.15	88.2
Memory: 4 GB Kingston	14	14
Hard drive: Seagate 2TB ST32000644NS	8.33	7.91
others	20.62	N/A
Total	190.1	111

in Fig. 3, they can be converged into two lines with little difference. In other words, the disk's power consumption is insensitive to the workload charactersitcs comparing to CPU.

Thus, the original model in Eq. (2) has to be modified to reflect the above findings. To estimate the power cost of a query plan, we are essentially interested in the **marginal power** it consumes. As the CPU is the only one that contributes significantly to active power, we have to focus on the cost of processing tuples and index tuples in CPU instead of I/O operations. In other words, we have

$$P = W_{cpu} \times \frac{N_{tuples}}{T} + W_{cpu\_index} \times \frac{N_{indexed\_tuples}}{T} \tag{3}$$

Other components are ignored because their contribution to the active power is negligible. Note that the above model is significantly different from what is used in a traditional query optimizer with query processing time (or throughput) as the optimization goal [16, 26]. In the latter, the I/O cost is the dominating factor that often overshadows CPU cost.

4.2 Observations on Workload Characteristics

To establish a safe zone for observations, we define the steady state in the power curve of the CPU usage when executing different workloads with the multiprogramming degree. In the experiment setup for identifying the power cost for each relational operators, we hacked the kernel in the DBMS to generate specified execution plan for each relational operators, such as sequential scan. The first step to verify those cost comes from verifying power cost of queries running using each relational operators only, without interference of other workload noise, such as other queries. Thus, for identifying the power cost for each relational operators as well as the power profiles for those workloads, we define our detection steady state as shown in Fig. 4. It is the period with only one query running in the system and the power cost output is stable.

The model shown in Eq. (2) and Eq. (3) reflects the intuition that the total *energy* consumption of a query plan is proportional to the "work" (i.e., number of operations such as $N_{indexed\_tuples}$ and N_{tuples}) to be done by that plan. Therefore, power consumption is related to the work intensity (i.e., work per unit time) instead of work. For the same

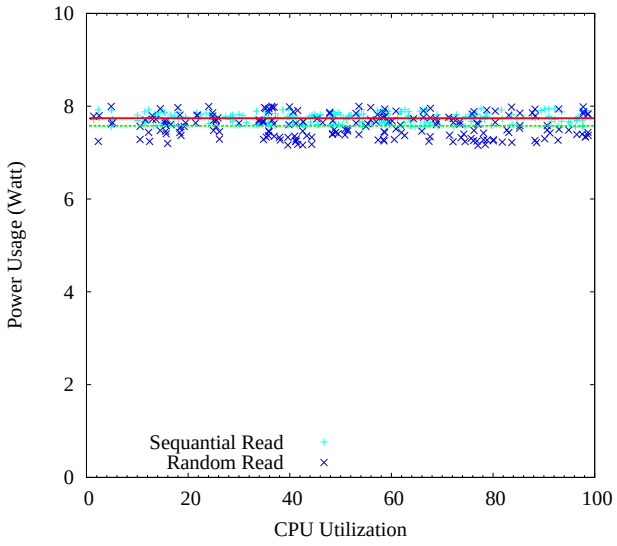

Figure 3: hard disk power consumption under sequential read and random read workloads

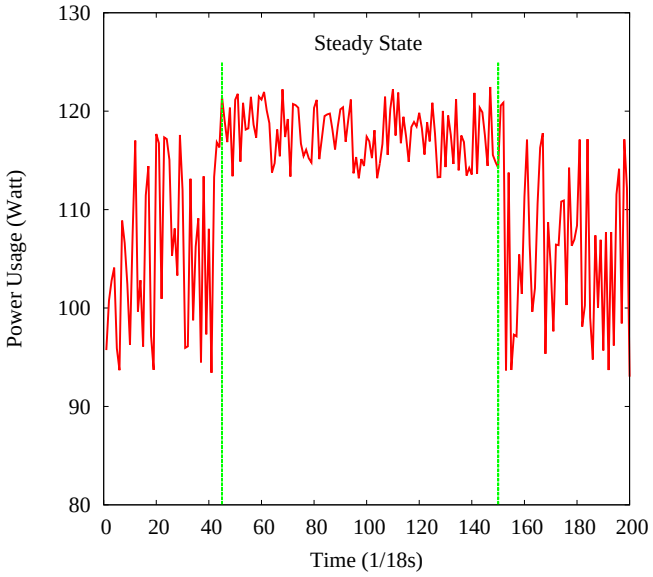

Figure 4: The steady state for power measurement

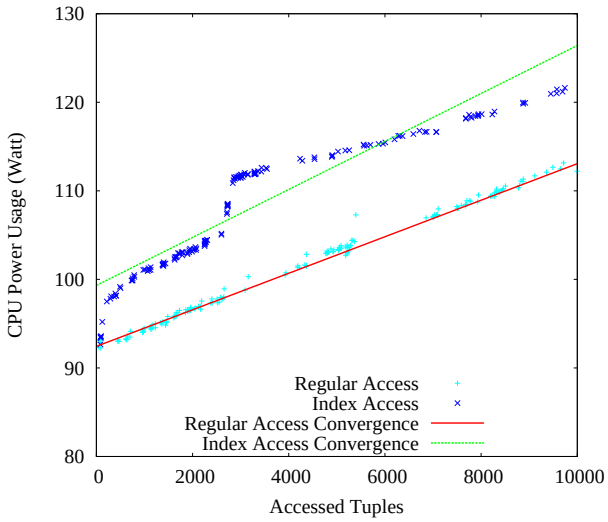

Figure 5: CPU power consumption under different numbers of tuples accessed

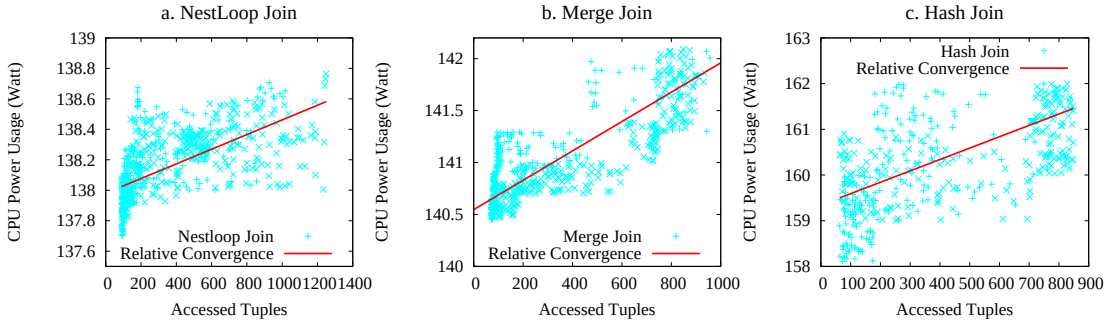

Figure 6: CPU power consumption of Join algorithms under different numbers of tuples accessed

reason, W_{cpu} and W_{cpu_index} represent the energy consumption per operation. Again, the same intuition was adopted in query plan evaluation in a traditional query optimizer.

The results of an extensive set of experiments, however, show that the actual *power* consumption of a database server depends directly on the number of database operations. In such experiments, we run the same query multiple times. By controlling the query parameters (e.g., range of search conditions) or size of the underlying database tables, the number of tuples accessed by the query processing algorithms changes in different runs of the query. For example, Fig. 5 shows the CPU power consumption of such runs for two queries: one with a sequential table scan and the other with indexed table scan. We can clearly see that for both queries, the CPU power increases monotonically with the total number of tuples accessed. We have tried such experiments for single join queries with two size group queries and similar trends were observed as shown in Fig. 6.

We believe the reasons for the above observations (which is somehow counterintuitive) are complicated. Our explanation is: reading larger number of tuples from a file needs more processing overhead such as updating the free space map and visibility map processes in DBMS, swapping and scheduling processes at the OS level. Such overhead translates into extra CPU power cost for reading larger number of tuples. Although an in-depth investigation is needed as future work, we believe we have enough evidence to modify the previous power model into:

$$P = W_{cpu} \times N_{tuples} + W_{cpu\_index} \times N_{indexed\_tuples} \tag{4}$$

The biggest difference is that we dropped the processing time T from Eq. (3), and the physical meaning of parameters W_{cpu} and W_{cpu_index} is the *marginal power cost of processing each relevant operation*. Note that the

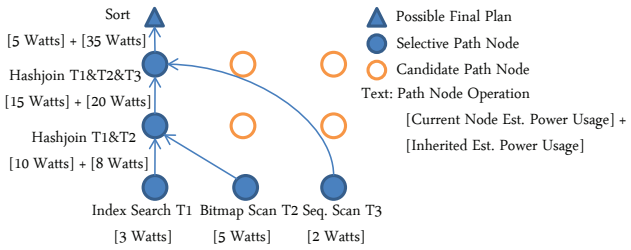

Figure 7: One example of power cost estimation in a query tree generated by PostgreSQL

slope of the two regression lines in Fig. 5 is a good indicator of values of such parameters.

Given Eq. (4), the task of power estimation is to quantify the numbers of basic operations N_{tuples} and $N_{indexed\_tuples}$, and the two parameters (i.e., unit power costs) W_{cpu} and W_{cpu_index}. With N_{tuples} and $N_{indexed\_tuples}$ being provided by the existing query optimizer, the key problem becomes to quantify the two model parameters W_{cpu} and $W_{cpu-index}$. In the following steps of modeling, we first treat both as static parameters via calibrating real power measurements of different workloads (Section 5). Realistically, the model parameters should be modified under different system states and workload features. For example, Fig. 2 shows that the marginal CPU power cost levels off when the CPU utilization increases - this means our model parameters should also be smaller when the CPU is heavily loaded.[3]

5 Static Modeling

In conventional DBMSs, the query optimizer precomputes the time cost for each relational operator in the plan and chooses the cheapest candidate as the final execution plan. As mentioned in our previous work [31], a power-aware DBMS can use a model to estimate the power cost of the query plan, and such estimations become inputs to a composite cost model that encompasses both power and time costs. A System R type query optimizer follows a bottom-up approach to build query trees, and alternative plans in each subtree (representing a relational operator, as shown in Fig. 7) will have to be evaluated. For that purpose, there is a need to estimate power cost at the relational operator level.

5.1 Cost Model for Relational Operators

The theoretical model shown in Eq. (4) is too general to capture the resource consumption patterns of individual relational operators, therefore we provide more refined models to describe costs of the most popular operators. 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. For simplicity in presenting the models, let us redefine a few key quantities in Table 2. Following that, a summary of the operator cost models is shown in Table 3. Note that values of a few quantities, including n, S and H, are provided by the existing query optimizer. The model parameters (e.g., w_i, w_s) are obtained as follows. We composed a set of simple queries that include single table scan and two-table joins based on the TPC-H data. At the same time, the query optimizer is hinted to execute those queries in each specified operation plans, such as sequential scan, no matter whether it is efficient or not. In this way, we collected a set of data and power usage statistics during query execution.. We feed the collected data and proposed model into the LP solver of the General Algebraic Modeling System (GAMS) software [1] to find the best values of coefficients to adjust the model to the statistics. After a few times' training process, the model is established to predict the power cost to executing relational operators. In the following paragraphs, we elaborate on the development of the models in Table 3.

The above models (for both single table and join operations) can be combined to form complex models for any arbitrary query plan. For example, the top node in Fig. 7 shows a total power cost of $50 + 350 = 400W$, in which 50W is the cost of sorting and 350W is that for child node representing a hash join of three tables.

[3]Otherwise, a plan with huge N_{tuples} or $N_{indexed\_tuples}$ values will carry an unreasonably high power tag.

Table 2: Key Quantities in Power Estimation Model.

Symbol	Definition
n	The number of tuples retrieved for CPU processing
w_s	The average CPU power cost for processing a regular tuple
w_i	The average CPU power cost for processing an index tuple
w	The average CPU power cost for sorting a tuple
τ	The power cost for sorting a group of tuples
H	The number of hash partitions

Table 3: Power cost functions for basic relational operators.

Methods	Cost function
Seq Scan	$w_s \times n$
Index Scan	$w_i \times n$
Sorting	$w \times n \log n$
Bitmap Scan	$w_i \times (1+\tau)n$
Nested Loop Join	$w_i \times n_1 + w_i \times n_2 \times n_1$
Merge Join	$w_i \times (n_1 + n_2 + \tau)$
Hash Join	$w_i \times (n_1 \times H + n_2)$

6 Static Model Validation

6.1 Results of Single Table Models

We generated a set of data files (w/wo indexes) for single table scans. Moreover, we prepared a set of similar queries (queries with the same structure but different selection predicates) to test the table scan models. For each scan operation, there are equal number of queries visiting large table files (e.g., 6GB *lineitem* table in TPC-H) and small table files (e.g., 10MB *order* table in TPC-H). Each experiment was repeated 100 times and we computed the average and plotted the results in Fig. 8.

As seen from Fig. 8, the sequential scan model's EER is less than 0.5% for both large and small file scans. We believe this is a high accuracy as some random error is inevitable, and errors lower than 0.5% are essentially negligible. Results for index scans (bitmap and regular index scan) are also very promising. The estimated power cost is nearly the same as real power measurement at all cases ($\pm$ 0.5 Watts). In summary, our static model works very well for single table operations by achieving accuracy that is over 99%. One other thing to point out is that the power consumption is always higher in cases of large file access than in small files, thus supporting our conclusion that power consumption depends on the amount of work to be done (Section 4).

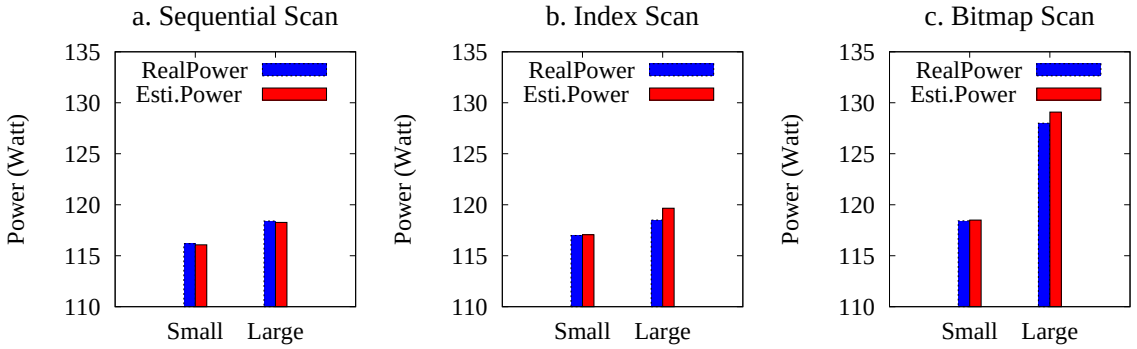

Figure 8: Empirical validation results of single table operations' power models.

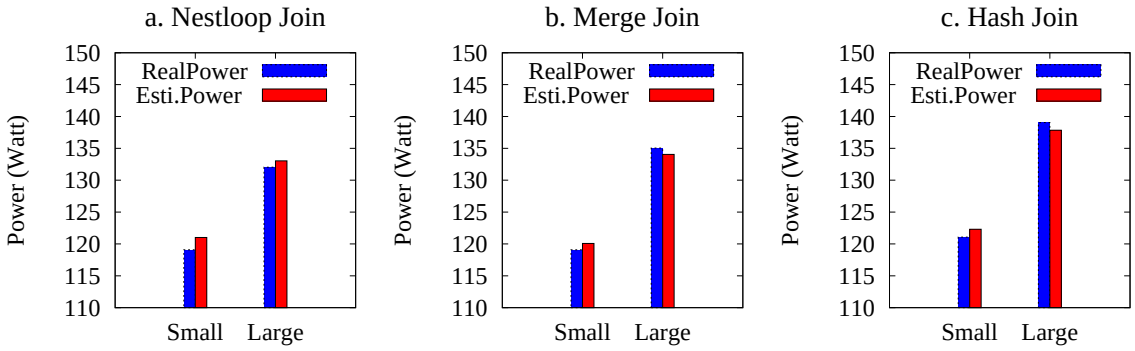

Figure 9: Empirical validation results of multi-table operations' power models.

6.2 Results of Multi-table(Join) Models

The verification of join power models uses the same experimental setup and data files in previous experiments (Section 6.1). From the TPC-H official tool, we created a set of queries with one single join of two tables. Each join operation is a combination of two scans and one join operation (see Fig. 7). It is not a surprise that system power increases much faster than scan operators when the size of the joined tables (thus resource usage) increases (Fig. 9). Again, the observed EERs are less than 0.5%, indicating the effectiveness of the join models.

6.3 SDSS Validation Experiments

To test our models under very large datasets, we implemented a database from the published release 7 SDSS data. Then we created three workloads: workload I is an equality search based on a sequential single table scan; workload II is a merge join of two partial tables after range searches; and workload III is a range search based on sequential scan. We run each single-query workload for 1,000 times with different search predicates generated randomly, and the results are shown in Fig. 10. The size of the largest table scanned in such queries was 2TB. As we can see, the difference between the estimated power and measured power is very small in almost all cases. This is another successful case for the static power models.

6.4 TPC-H Joins Validation Experiments

In the final set of experiments, we used composite workloads that contain a random subset of the TPC-H queries. According to Fig. 11, the raw EER in this experiment ranges from 4.07% to 10.67%. However, the high EERs are caused by inevitable noises in power measurement, as seen by the fluctuations in the green line. Thus, we took a moving average of the measured power (average power cost within a fixed time window) and the results are shown as the red line in Fig. 11) and compared it with the estimated power – the difference between the two are named modified EER (MEER). The largest MEER value is 2.65%. We also tried single-query workloads for all 22 queries of the TPC-H benchmark, and the estimation results are demonstrated in Fig. 12. Again, we observed very high accuracy, with an average MEER of 3.8% and the highest MEER reaching only 4.6%.

6.5 Limitations of the Models

The above experiments show that the power prediction is very successful in a static environment. However, our modeling task is far from complete. When we change workload features and system resource availability, our models can easily fail. We first test the system with 9 different composite workloads generated from TPC-H under an MPL that is greater than one. Specifically, we initialized 100 client threads and allowed queries sent from all clients run concurrently (i.e., MPL can reach 100). The results of such experiments are plotted in Fig. 13 – we can see that the EER for all workloads are over 40%, with the highest one reaching 65%. These are very inaccurate estimations as they almost reach the upper bound of possible errors, such bounds are described in Appendix 1. Note that all absolute errors in Fig. 13 are negative, meaning the model systematically underestimated the power cost of the workloads.

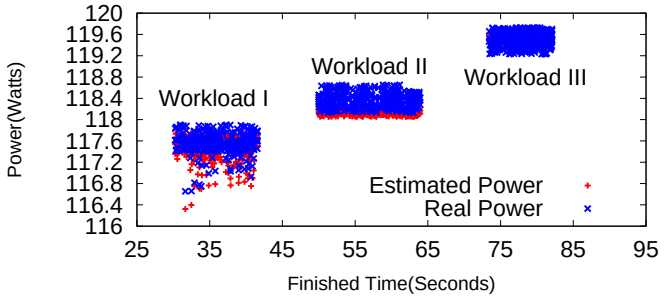

Figure 10: Power model testing in SDSS database using three different queries

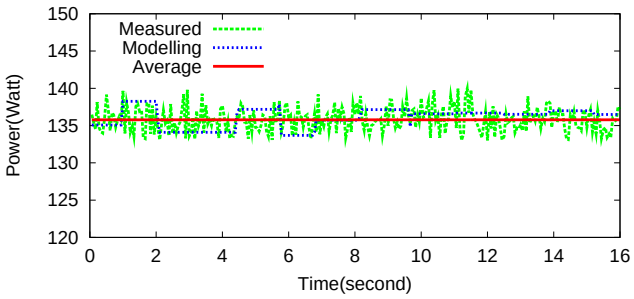

Figure 11: Power model validation using a batch workload in TPC-H benchmark

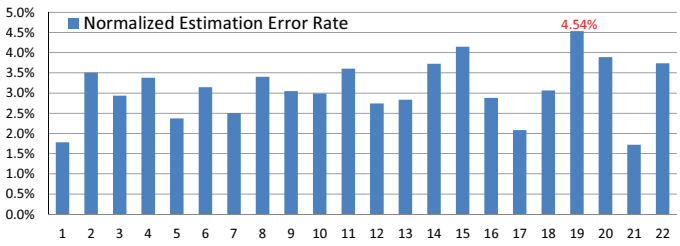

Figure 12: Power estimation errors upon running 22 queries in TPC-H

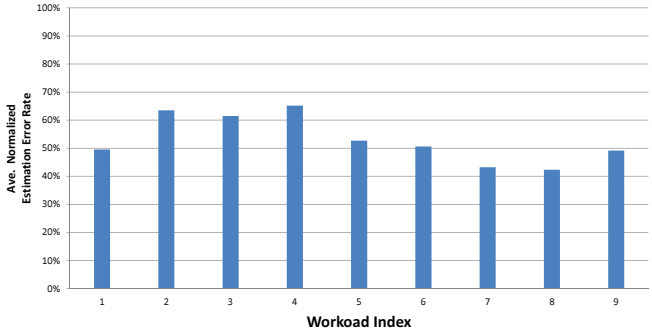

Figure 13: Power estimation error for a mixed TPC-H workload with nine different query sets

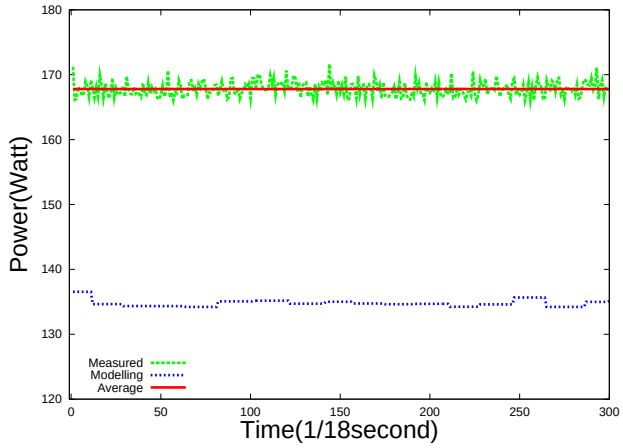

Figure 14: The operator estimation model fails under a change of system contention. The average ERR is as high as 65%.

In another experiment, we simulated a change of CPU resource availability by introducing CPU-intensive non-database jobs into the system. Specifically, we fork a process that creates a number of children threads to compute the Fibonacci sequence. Again, this caused a serious underestimation of power consumption (Fig. 14). This clearly shows that our model needs to be updated dynamically to capture the changes of system status. One might argue that the problem is caused only by the models' failure in capturing the correct baseline power of the system, and can be easily solved by reading in a baseline power in real-time. However, such a solution would not be robust at all. First, competition between concurrent queries has profound effects on power consumption. Second, when such effects are mixed with those caused by system states (such as that in Fig. 14), it is almost impossible to tell them apart.

6.6 Source of Errors

Many factors can contribute to the errors in power estimation of database queries. Roughly, such factors can be put into three categories.

- *System status*. Run time state change of the database system and even the OS can cause significant errors in power estimation. A simple example can be seen in Fig. 2: the marginal increase of CPU power is not linear to the CPU utilization. In other words, the parameters w_s and w_i should have different values under different system contention, and the change of the latter cannot be easily detected. Other events such as initialization of competing computational jobs can cause the same problem;

- *Operation quantity estimation error*. Errors are inevitable in estimating the number of input tuples (e.g., n_1, n_2 in Table 3) for a relational operator in the plan. In our models, such values are provided by the existing query optimizer. In fact, this is a classical problem in query optimization as the query processing time also depends on such quantities [9].

- *Workload dynamics*. A workload generally contains queries with different resource consumption patterns and the interactions among concurrent queries are very complex. Workload features may change over time and significantly impact power cost by changing variables such as cache hit rate and concurrency level.

We mark those error sources in the system shown in Fig. 1. 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 it is impossible to locate all the possible factors, and model their effects on power consumption. The solution we propose and implement here is to use an online feedback model that adjusts the weight parameters of each relational operator's estimation function by capturing the recent trends of system and workload dynamics in order to decease the EER.

7 Dynamic Modeling

In this paper, we propose an *online model estimation* strategy to minimize the errors. The main idea is: we keep the structure of the previous physical model we develop. We then treat the database system as a black-box and model the cost parameters (w_s and w_i) in our existing model as system-level variables whose values reflect the combined effects of all possible system/environmental factors. We then use a feedback control based mechanism to periodically update the parameters using real-time power measurements. This means that, even errors in estimating the operator quantity will be compensated for by adjusting such parameters.

We modified the power consumption of each relational operator as in Table 3 into a period functions in order to adjust itself with the dynamic system environment. Let's introduce some notations first. We have a checking period T that online model will automatically check the system dynamics such as CPU utilization, multi-thread degree and etc. in order to get the parameters' (positive or negative) gains and power baseline before predict one query's running power to refine the estimation results. The operation vector $\vec{n_{kj}}(i) = \{n_1, n_2, \cdots, n_j\}$ stands for the relational operators that are used to execute query k during period i. We will talk about the online parameter estimator in the next section.

In the experiment system, the estimated power parameters W are based on our previous data mining results used in the static model. We conduct power statistics identification experiments for all the TPC benchmarks that we have and collect all the result data. Computed by GAMS, we gained the initial weight vector $\vec{w} = \{0.1421, 0.3221\}$. The performance turns out to be very good in our test set as shown in Fig. 12. The initial value of the baseline power is 110 Watt, the idle power that we have identified as in Table 2. Now we know how to refine the baseline power to estimate total energy cost. However, the problem is how we could use this data to adjust the model to accurately estimate the online power cost of each query.

7.1 Online Model Design

In this section, let us introduce the online model estimation scheme, which is based on the a refined Recursive Least Square (RLS) estimator [30, 28, 29] with directional forgetting according to our preference.

We will maintain an operation vector $\vec{n} = \{n_1, n_2, \cdots, n_m\}$ to hold quantities of all operations of queries currently running in the system. Recall that the values in $\vec{n}$ are provided by the query optimizer. In general, let $\vec{w} = \{w_1, w_2, \cdots, w_n\}$ be all the parameters to be updated and $k = \sum_{j=1}^{n} w_j$. In our case, we have $\vec{w} = \{w_1, w_2\}$ and $k = w_1 + w_2$. We then define a new vector $\vec{w'} = \{w_1, w_2, k\}$, and denote the value of $\vec{w'}$ at period j as $W(j)$. Similarly, for the operation vector $\vec{n} = \{n_1, n_2, \cdots, n_m\}$, we define another vector $\vec{n'} = \{n_1, n_2, \cdots, n_m, 1\}$ and denote the value of $\vec{n'}$ at period j as $N(j)$. At each period, the actual power consumption of the server, p, is measured. The RLS model generates a quantity $p(j)$ as the baseline power from the measurements of the last $j-1$ periods and current p as follows:

$$p(j) = \frac{((j-1)p(j-1) + p + P_I)}{j} \tag{5}$$

where P_I is idle power for all hardware components except the CPU, which is $111 - 88.2 = 22.8W$ in the server we used according to Table 2. We take the system idle power as the initial value of $p(j)$, i.e., we have $p(0) = 111W$.

The baseline power defined above is used to update the parameter vector $W(j)$ as follows:

$$W(j) = W(j-1) + \frac{e(j)N^T(j)P(j-1)}{\lambda + N(j)P(j-1)N^T(j)} \tag{6}$$

where $e(j) = p(j) - N^T(j)W'(j)$ is the estimation error, $P(j-1)$ is the covariance matrix of vector $N(j)$, and λ is the constant forgetting factor within $[0, 1]$ – a smaller λ enables the estimator to forget the history faster. The following routines are invoked at the beginning of every period j of model updating: (1) the RLS estimator records the operator vector, $N(j)$ and calculates baseline power $p(j)$; (2) it computes $W(j)$ according to Eq. (6).

The RLS estimator adapts itself so that $e(j)$ is minimized in the mean-square sense. When the two variables, $\vec{n}$ and $p(j)$, are jointly stationary, this algorithm converges to a set of tap-weights which, on average, are equal to the Wiener-Hopf solution [20]. As a recursive algorithm, the RLS estimator has very low computational overhead (tens of microseconds as we recorded in our experiments). It is also robust against different workloads and system conditions.

One special note about $p(j)$ is: instead of being measured via a power meter, p is translated from CPU utilization (provided by the OS) following the regression curve shown in Fig. 2. The advantages of this method are: (1) real measurements from a power meter are always associated with delays from the communication channels and resource

overhead; (2) our database system can be deployed in a server without a power meter connected - they are only required for generating the initial values for the model parameters. This is a huge benefit in data centers that host a large number of servers. We also compared this method with real power measurements and found that the differences are negligible.

7.2 Online Model Validation

The experiments are setup using the same hardware environment as mentioned in Section 3. However, our enhanced model tends to solve more complex problems. Thus, the workload generator will produce dataset and workload that create extreme cases which our static model may not handle well.

The workload generator supports three sets of benchmarks: TPC-H, TPC-C and SDSS. (1) The TPC-C benchmark tools is a close environment to verify our model's performance as a black box test. (2) With the 22 queries extracted from TPC-H and many other similar ones (modified from the standard 22 queries with different constrains), the generator could produce a TPC-H scheme workload that consists of queries with a predefined distribution of arrival time and features like resource sharing (join the same table or within the same selectivity histogram within the same table), priority and multiprocessing degree. (3) In the experiment, we have one TB scientific database SDSS in order to obtain our model's performance when facing a large database. It includes 53 million unique objects like galaxies quasars, stars and unknown classes (Although we don't quite mind what content the data set has). The queries we created for this data set are large table scan and limited number of joins (mostly two table joins). The purpose of this workload is to identify the model's extension to estimation energy cost when there is not too much changed in the time estimation.

Also, we have predefined two kinds of workload sets with contrary features. The purpose to design those special test sets is to maximize the relevant error factors which we discussed in section 6.6, check whether our new model can handle them and how much performance it can achieve.

7.2.1 Type I: Fine-grain Parallel Workload v.s. Coarse-grain Parallel Workload

Coarse-grained parallel workload contains queries of large computations, lower interaction rates and little data shared. While on the other hand, fine-grained parallel workload is generated by queries of small computation, large interaction and considerate number of shared information. The purpose of introducing this experiment set is to verify our model's sustainability in low & high data sharing circumstance. Because in a barely shared system environment, the system dynamic changes rapidly since new resource reallocation and other preprocessing operations done from OS perspective. This result will provide us upper and lower bound of estimation error rate(EER) of our model with RLS technique when handles unpredictable system dynamics.

7.2.2 Type II: Data Intensive Workload v.s. Random-Selective Workload

As we discovered from EER in resource estimation, the major reason for this failure is because the missing information of the data histogram of tables. However, to verify that, the workload generator will produce a special purpose test set that contains (1) queries that will visit similar part of the data histogram from time to time so that the statistic model will be barely updated and precise since those information will be captured in the first or second query execution. (2)queries that follows a randomized access behavior to visit all the histogram of the data table and the data table will be update periodically by some other queries. The latter workload requires the optimizer to be quite sensitive with the updating so that it will capture the dynamics to adjust the data histogram statistic for better prediction. In all, the purpose of this workload is testing our mechanism in solving the resource estimation, more specifically, selectivity estimation problem.

7.2.3 Results of Type I Workloads

In such experiments, we created query pools for each of the type I workload sets following the descriptions in Section 7.2 and randomly picked queries from such pools. To be specific, we tried 9 workloads with query set sizes ranging from 10 to 2,000 for each of the four workload sets. As shown in Fig. 15 and Table 4, the MEERs generated by the RLS models are significantly smaller as compared to those under the static models. Also, by comparing the results of two workloads under RLS model, we can see that our model handles the share-everything (fine-grained parallel) workload as well as the share-nothing (coarse-grained parallel) workload, with an average MEER of 11.55%

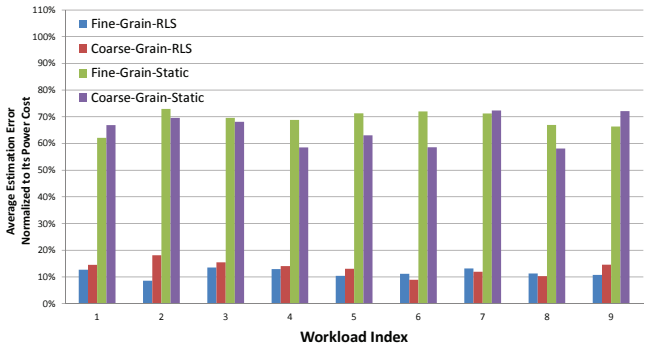

Figure 15: The EER comparison of parallel workload sets under different models.

and 13.42%, respectively. The fact that both query patterns show similar model behavior shows that our model can effectively handle the interactions among queries.

Table 4: Average MEER for two workload sets using RLS model and static model. FG stands for fine-grained parallel workload and CG stands for coarse-grained parallel workload

Queryset Size	FG-RLS	CG-RLS	FG-Static	CG-Static
10	12.79%	14.80%	60.46%	66.47%
50	8.50%	17.70%	72.60%	69.31%
100	13.48%	15.25%	69.30%	68.60%
200	12.60%	14.24%	69.81%	57.68%
300	10.27%	12.86%	71.35%	62.69%
400	11.09%	9.13%	71.24%	59.63%
500	13.20%	11.72%	72.21%	73.09%
1000	11.15%	10.20%	65.00%	56.44%
2000	10.90%	14.87%	67.32%	71.77%

In a concurrent system as DBMS, studying and defining the model performance with the parallel workloads are necessary. As we observed from our experiment, shown in Fig. 15 and Table 4, the EER decreases significantly compared with the static model. Also, comparing the result of two workload query sets under RLS model, in general, our model handle the sharing-everything(fine-parallel) pattern the same as sharing-nothing(coarse-parallel) workload pattern. The average absolute EER is about 20.3%. The reason why both data pattern shows similar power estimation behavior is that their difference result in different number of operations in memory and disks (coarse-parallel workload may require more memory, and demanding pages from disks) that contribute barely in the active power. So, in the context of power estimation, this workload dynamics won't affect our estimation precision.

7.2.4 Results of Type II Workloads

As shown in Fig. 16, the model shows a pretty good behavior when handling with data intensive case. The query always visit the same part of the table, which is in the memory according the DBMS's memory management policy. Thus, the workload is running under a approximately steady system environment. That is also the reason why the static model for this workload shows a uniform-like estimation behavior. On the other hand, when it comes to random access data behavior, since the data's probability histogram is not a uniform distribution, sometimes it could cause huge estimation error when the system is waiting for I/O but the estimator cannot capture the change. On average, the RLS model handles randomize access case well, which at some points, handles the resource estimation error in an indirect way.

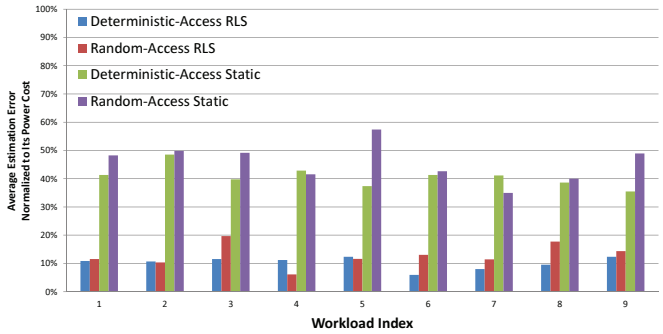

Figure 16: The EER comparison of different data-access-manner workload sets under different models.

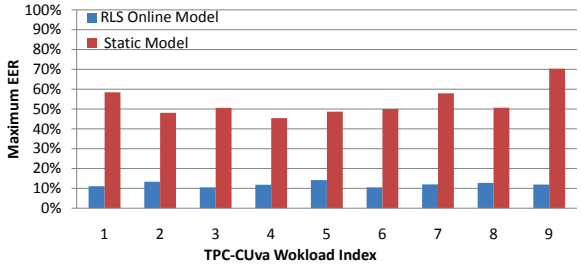

Figure 17: The EER of running an unknown TPC-C workload

7.2.5 Black Box Validation

We also validated our model within a closed query environment generated from a non-commercial TPC-C tool called TPC-C UVa (http://www.infor.uva.es/˜diego/tpcc-uva.html). Since we cannot change any distribution or the query composition inside the workload, this serves as a perfect tool for black box testing. As seen in Figure 17, the average EER of RLS-based model is around 10% in a 10GB DBMS configuration compared to the static model's EER of 51.4%. Note that we recorded the maximal error in Figure 17 - the average errors are even smaller. This clearly shows that our RLS model is robust even under a workload whose internal features are unknown to us.

7.2.6 The Effects of Forget Factor λ

As seen in Fig. 18, the forgetting factor λ could affect the accuracy of the RLS model significantly. For the SDSS workloads, the results are stable without showing much difference under different λ. Our explanation is: most queries in SDSS are waiting for I/O due to the sheer size of the table - this creates a very static environment. However, in a dynamic environment, if the system states follow a historical trend (e.g., those of coarse-grained parallel workload or

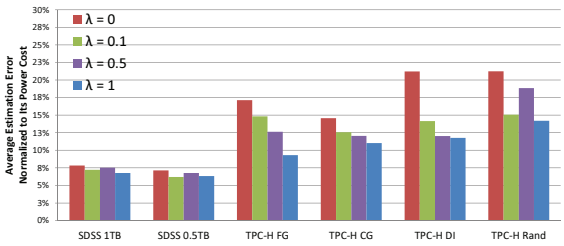

Figure 18: The EER comparison of different workload sets under λ configuration in the RLS model

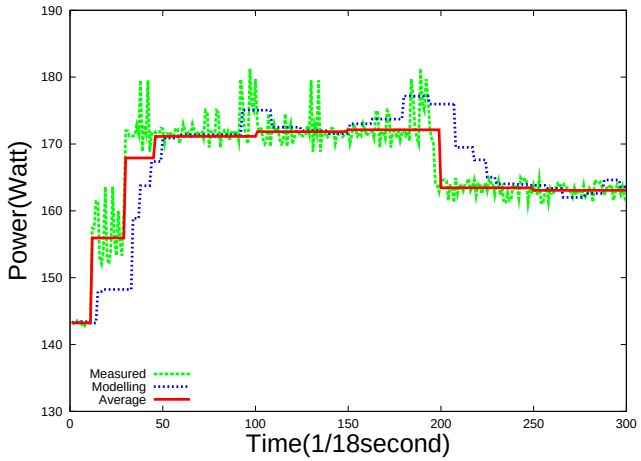

Figure 19: Model behavior at runtime upon sudden changes of DVFS level

a deterministic access workload), the increasing value of λ will gain significant benefits in terms of model accuracy.

7.2.7 Tests Under System Dynamics

In this experiment, we simulated sudden changes of the system states by changing the CPU power consumption pattern at runtime. We took advantage of the Dynamic Voltage Frequency Scaling (DVFS) feature of the CPU to implement that. In Fig 19, we present a sequence of power data collected "on-the-fly" by marking the measured power as the green line, and the estimated power as the blue line. At first the system is running under 70% frequency, with a power consumption of the system around 143 Watts when running a mixed TPC-H workload. In about one second, the CPU frequency increases to 80%, and then jumps to 90% after another second. Our model started to adjust the parameters to catch the changes of CPU frequency right after the first jump, and managed to minimize the estimation error at about one second after the second jump. The system experienced another sudden decrease of power at the 11th second, and the RLS model, again, is able to respond to that in about two seconds' time. The average MEER we calculated during this whole period is about 9.72%. This clearly shows our model's ability to deal with system dynamics.

8 Conclusion and Future Work

In this paper, we have presented how we build up an accurate cost model for estimating power costs of query plans in the query optimizer. Also, we describe how the general static model could fail due to the dynamic environment and unpredictable workload characteristics. In the strong assumptions we use to build this model as described in 3, we achieve an accuracy of 95.6%, on average, on the power cost estimation. Still, we believe that the result could be even better if we can remove the CPU power noise from it. Also, we present the extension of the model in order to adjust itself to the dynamic environment which makes the model more feasible to other environment such as multi-programming and different distribution of workloads. Also, a promising future research direction in this area is the consideration of the cost power model of other operations such as insert and massive update. Also, how to extend the model in a distributed environment is an interesting direction.

References

[1] The general algebraic modeling system (gams), 2011.

[2] F. Ahmad and T. N. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. *SIGARCH Comput. Archit. News*, 38(1):243–256, 2010.

[3] M. M. Astrahan, H. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System r: Relational approach to database management. *ACM Transactions on Database Systems*, 1:97–137, 1976.

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

[5] A. Berl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. de Meer, M. Q. Dang, and K. Pentikousis. Energy-efficient cloud computing. *Comput. J.*, 53(7):1045–1051, 2010.

[6] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case for power management in web servers, 2002.

[7] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proc. of HPCA*, 2001.

[8] Charles Lefurgy. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, 2003.

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

[10] Y. Chen, A. Ganapathi, and R. H. Katz. To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency. In *Green Networking*, pages 23–28, 2010.

[11] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM Trans. Database Syst.*, 9(2):163–186, 1984.

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

[13] S. Irani, G. Singh, S. K. Shukla, and R. K. Gupta. An overview of competitive and adversarial approaches to designing dynamic power management strategies. *IEEE Transaction on VLSI systems*, 13(12), 2005.

[14] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *SoCC*, pages 39–50, 2010.

[15] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. S. Owicki. Competitive randomized algorithms for non-uniform problems. In *SODA*, pages 301–309, 1990.

[16] R. Kooi. *The Optimization of Queries in Relational Databases*. PhD thesis, 1980.

[17] P. Kurp. Green computing. *Commun. ACM*, 51:11–13, October 2008.

[18] W. Lang, R. Kandhan, and J. M. Patel. Rethinking query processing for energy efficiency: Slowing down to win the race. *IEEE Data Eng. Bull.*, 34(1):12–23, 2011.

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

[20] J. B. Lawrie. A brief historical perspective of the wiener–hopf technique, 2007.

[21] Y.-H. Lu, L. Benini, and G. D. Micheli. Operating-system directed power reduction. In *ISLPED*, pages 37–42, 2000.

[22] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *VLDB*, pages 149–159, 1986.

[23] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy optimization for dynamic power management. In *Design Automation Conference*, pages 182–187. ACM Press, 1998.

[24] 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.

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

[26] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD Conference*, pages 23–34, 1979.

[27] 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*, SIGMOD '10, pages 231–242. ACM, 2010.

[28] 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*, 43(1):31–55(25), 2005.

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

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

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

[32] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: managing energy as a first class operating system resource. In *Proc. of ASPLOS*, 2002.

Appendix

1 EER Worst Case Analysis

When analyzing the sources of errors, we also ask ourselves another question: *compared with time estimation, is there a analyzable upper bound of EER in the worst?* The answer is yes. Given our experiment setup for an example, its maximum power cost is about 190 Watt and idle power is about 110 Watt when the CPU utilization around 3% to 10% (shown in Fig. 2). Thus, consider two extreme cases: (1) estimator tells us that the system is running the query when the system is fully loaded, but in fact there is nothing running in the system at all. Using Equ. 1, EER is $(110 - 190)/110 = -72.7\%$. Clearly, the result is overestimated; (2) on the other hand, estimator says the system is running in the idle state, but actually the system reaches its maximum capacity. The error rate is $(190-110)/190 = 42.1\%$. The estimator is underestimated. Thus, in power estimation, unlike time estimation which may result unexpected unbound estimation error, the lower and upper bound of prediction is $[-72.7\%, 42.1\%]$ under our hardware configuration.

2 Best Linear Unbiased Estimator

In statistics, the rule for calculating the estimate of a given quantity based on observed data is an estimator. In our scenario, those are point estimators yield to single vector results. The best linear unbiased estimator, as its name called, is trying to provide us a best candidate of the estimate(s) of a linear model in a unbiased way. A linear regression model in which the errors have expectation zero and are uncorrelated and have equal variance, BLUE of the coefficients is given by the ordinary least squares estimator. In our linear mixed power cost estimation models for the estimation of the power cost under random effects, we also have the BLUE used for the best estimation of the power cost coefficients by using least square estimator. BLUE is mainly utilized in the model to talk about the estimating fixed effects and sometimes random effects. In our practice, the parameters are defined to associated with random effect terms, which are unknown. They are the variance of the resources data and residuals. Simply plugging in the estimated parameters into the predictor, without accounting additional variability, leads us to overly optimistic prediction variances for our model.

3 Assumptions

[11] gives assumptions for time estimation models at early stage of database research. Similar as their work, we list some assumptions as the basis for building our static model. following the conventional discussion about the content of the DBMS profile.

1. Uniformity of unit power cost for processing one tuple/indexed tuple: the power usage for processing one tuple/indexed tuple in CPU is always the same.
2. Constant number of tuples per page: the probability of referencing any page is $1/P$, where P is the number of pages.
3. Random replacement of tuples among pages: the probability of referencing any tuple is $1/T$, where T is the number of pages.

Assumptions 1 affects whether exists power coefficients in the power cost metrics. Assumption 2 and 3 affect the number of estimated resource (e.g. the cardinality of data table). Those assumptions decrease the accuracy of estimation but gain us great help at starting line without worrying about complexity of the system and workloads. When we started to build online model based on the static model, the latter two assumptions are loosened in order to enhance the estimation performance.