Characterizing Peak Power Behavior of Database Engines

Mayuresh Kunjir   Puneet Birwa   Jayant R. Haritsa

Technical Report
TR-2012-01

Database Systems Lab
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560012, India
`http://dsl.serc.iisc.ernet.in`

Abstract

Database engines often consume significant power during query processing activities, motivating researchers to investigate the redesign of their internals to minimize these overheads. While the prior literature has dealt exclusively with average power considerations, our focus here is on peak power consumption. We begin by profiling the peak power behavior of a representative suite of popular commercial database engines in benchmark query processing environments, and demonstrate that their consumption can often be substantial. Then, we develop a pipeline-based model of query execution plans that lends itself to accurately estimating peak power consumption, suggesting its gainful employment in server design and capacity planning. More potently, given a space of competing plan choices, it could help identify plans with attractive tradeoffs between peak-power and time-efficiency considerations, and we present sample instances of such tradeoffs. Finally, we discuss extensions of our modeling approach to inductive pipelines and multi-query workloads.

1 Introduction

In recent times, addressing the power consumption incurred by computational hardware and software has become an active area of research, fueled by technological advances, environmental concerns and mobility considerations. As a case in point, database engines, a key component of many enterprise information systems, have been found to be major power consumers during their complex data processing activities. This led the 2008 Claremont report on database research directions to declare *"designing power-aware DBMSs that limit energy costs without sacrificing scalability"* as an important research area [1].

In assessing power utilization, there are two aspects – *average power* and *peak power* – that are of interest. Average power consumption impacts concerns such as long-term energy expenses and design of heat dissipation systems. Peak power consumption, on the other hand, is of relevance in server design, capacity planning, and prevention of overheating surges. In particular, it is mentioned in [6] that *"since cooling and power supplies are designed to accommodate peak consumption, reducing this overhead mitigates power and cooling limitations"*.

The prior literature on power consumption in database engines (covered in Section 7) has exclusively focused on average power considerations (e.g. [23, 19]). In this paper, we turn our attention to profiling, modeling and mitigating the *peak power* characteristics of database engines. While generic hardware mechanisms, such as voltage scaling [5], have been developed to manage peak power usage, they may not be compatible with the specific functional and performance expectations of the software packages executing on the system. Therefore it is important to investigate avenues for explicitly making database engines power-aware such that they work well even when constrained by a peak power budget.

Significant new challenges confront us when characterizing peak power behavior, as compared to average power models – in particular, we have to now (a) explicitly account for the *parallelism* of operators, as peak power represents the maximum aggregate consumption of concurrent operations; and (b) capture *bursty* or short-term phenomena during the course of a query's execution.

Peak Power Characteristics. We begin our study by profiling, on a well-provisioned workstation, the peak power behavior of a representative set of three state-of-the-art commercial database engines on query workloads sourced from the TPC-DS data warehousing benchmark [27]. Our experiments demonstrate that the power consumption incurred by such query processing can often take up a substantial fraction of the machine's dynamic power range, even when the queries are executed in isolation.

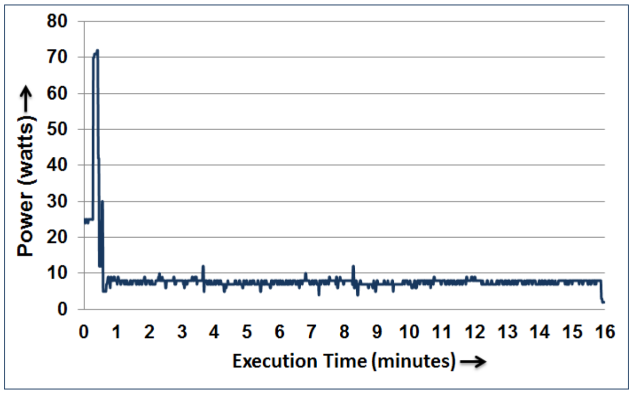
Further, there are often significant differences in the peak power consumption of the various engines – as a case in point, for Query 8 of the benchmark, two of the engines utilize around 30 watts of peak power, whereas the third engine consumes over 70 watts! This heavy usage lasts for a short initial burst of about 9 seconds, as shown in Figure 1(a), which tracks the engine's power consumption over the query's 16 minute lifetime. Its source can be traced back to the "pipeline" (sequence of concurrently executing plan operators) segment highlighted in the execution plan tree shown in Figure 1(b).

Regression Model. Motivated by the above empirical observations, we investigate whether it is feasible to *a priori* estimate the peak power consumption of a query. In particular, we look into whether this estimation could be carried out solely using information provided by the query execution plan, without requiring any run-time inputs. The challenge here, as mentioned earlier, is that multiple operators may be executing in parallel, especially on today's multi-core computing platforms, and we need to capture their *aggregate* power utilization. Further, in pipelined plans, power consumption of an operator is *dependent* on the maximum rate at which upstream operators are funneling data into the pipeline. Based on these observations, we have developed a model wherein a query plan is first segmented into pipelines, using techniques developed previously for SQL execution progress indicators [3, 10]. For each of these pipelines, we apply a mathematical function that takes as input the rates and sizes of the data flowing through the pipeline operators, and outputs an estimate of the peak power consumption. The function has been developed through fitting step-wise linear regression models [21, 22] on a set of training examples, which are carefully chosen with a view to minimizing the number of samples required to achieve the desired accuracy. Our evaluation indicates that, when the plan statistics are accurately estimated in the database system, this power model, albeit high-level, is typically able to estimate the peak power within $\pm$ 15% of the consumption encountered at run-time. Therefore, it appears to be a useful tool for incorporation in the design workbench of database servers.

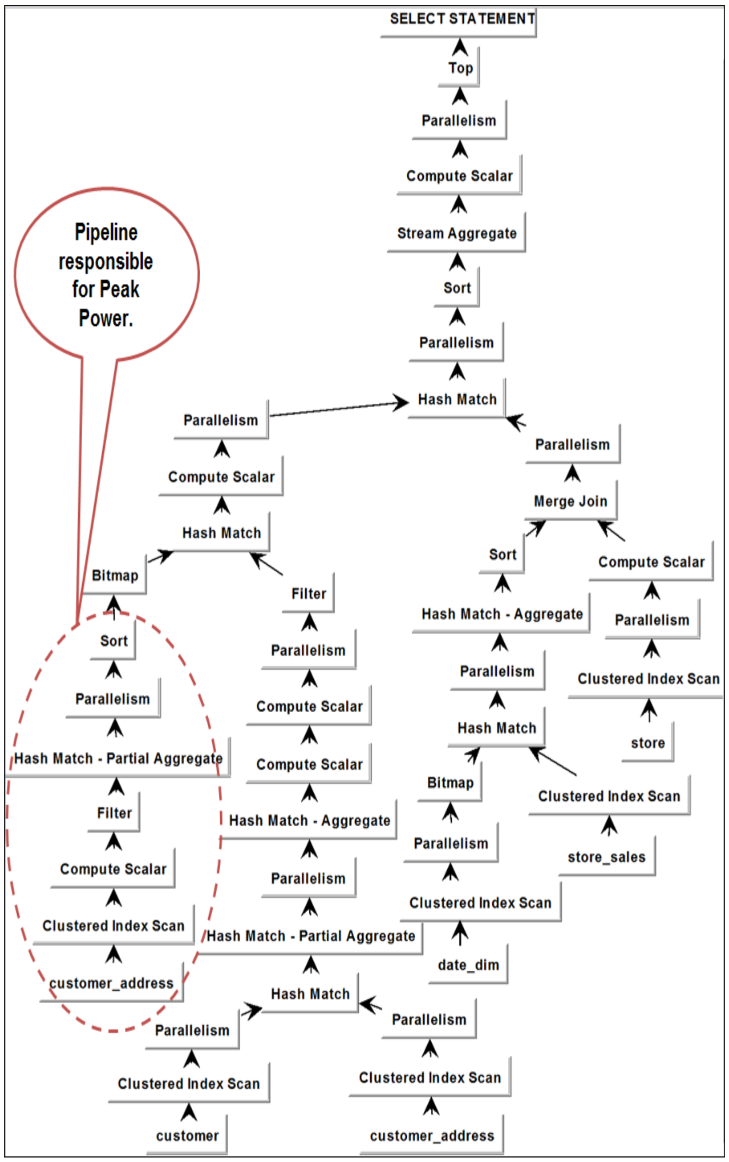
Query Plan Selection. Modern database engines typically choose query execution plans with the objective of minimizing the estimated query execution time, and to our knowledge, peak power considerations are currently not directly taken into account. In this scenario, it is entirely possible that peak power-efficient plans may be discarded in favor of time-efficient plans. A potentially potent application of the above-mentioned model is that it can help to quantify the peak power-efficiency of the various plan alternatives considered by the optimizer, thereby supporting making weighted choices between peak power and response time considerations. Our exploratory experiments in this regard, using candidates sourced from a parametric-optimal set of plans (POSP) [9], discovered, for some queries, plans that reduced the peak power by around *20 to 40 watts*. This is a significant reduction given the 80 W dynamic power range of our testbed machine. Further, these improvements were obtained even while confining our attention to only the subset of plans whose running times were within a factor of two of the optimizer's original time-efficient choice.

Black Box Environment. An important point to note here is that we are not privy to the internals of the commercial database systems. Therefore, our study has treated these systems as "black boxes", utilizing only their API functions. This means that our attribution of plan operator activity to the temporal power behavior in the training examples is perforce a coarse association. However, vendor design groups with access to engine internals could establish the correspondence more precisely, leading to improved peak power estimates. Further, it would be feasible to consider power-efficient replacement plans directly from the native plan search space, rather than our restricted POSP space.

Contributions. In summary, we demonstrate a first-cut proof-of-concept in this report that a viable



(a) **Temporal Power Behavior**



(b) **Execution Plan**

Figure 1: Power profile of TPC-DS Query 8

methodology can be developed for predicting the peak power consumption incurred by complex query processing on current database engines. Further, that opportunities exist to identify alternative query execution plans providing attractive tradeoffs with regard to peak power and response time. We also show that these plans are different than those identified for average power efficiency, highlighting the need for considering afresh the peak power problem. To the best of our knowledge, these results represent the first peak power characterization of database query processing, taking another step towards the ultimate objective of designing "green" database systems.

We hasten to add that all of the above comes, of course, with the implicit assumption that query processing is indeed the primary source of peak power consumption in database engines, as compared to other heavy-duty activities (e.g. nightly backup).

Organization. The rest of this report is organized as follows: In Section 2, we profile the peak power performance of commercial database engines on the TPC-DS benchmark. The pipeline-based model for identifying power-hungry segments of query execution plans is presented in Section 3. The robustness of model to changes in data and system environments is covered in Section 4. Then, in Section 5, we demonstrate instances wherein power-hungry plan choices can be replaced by comparatively power-efficient plans without incurring an excessive increase in execution times. Interesting modeling extensions are discussed in Section 6, while related literature is reviewed in Section 7. Finally, in Section 8, we summarize our conclusions.

2 Peak Power Profiles

In this section, we profile the peak power behavior of three popular commercial relational DBMS, operating in the TPC-DS benchmark environment. These engines are anonymously referred to as *EngineA* , *EngineB* and *EngineC* in the sequel.

2.1 Experimental Environment

Our experiments are conducted on a Sun Ultra 24 workstation configured with an Intel Core 2 Extreme Quad Core 3 GHz processor, 8 GB RAM, four 300 GB SAS hard disks (15K RPM), and running the 64-bit Windows Vista Business operating system. A Scale 1 (100 GB) version of the TPC-DS benchmark is used to populate the database.

2.1.1 Query Workload

From the 99 SQL queries that comprise the TPC-DS benchmark, we present results here for an illustrative subset of 16 queries. Their choice was motivated by the categorization in [15], wherein queries are classified based on their coverage of fact and dimension tables in the data warehouse schema. The classification is as follows:

Query Category	Number
Dimension Tables only	6
Single fact table	54
Multiple fact tables with sub-query joins	22
Multiple fact tables with sub-query unions	17

Our chosen queries include one from the first category (Q41), nine from the second (Q8, Q16, Q24, Q57, Q59, Q61, Q82, Q88, Q98), three from the third (Q58, Q64, Q83), and three from the fourth (Q49, Q66, Q76). These queries cover a wide spectrum of SQL features ranging from Aggregate functions to CASE statements.

2.1.2 Memory Management

For each database engine, the assigned memory is set to the same value, namely 6 GB of the 8 GB physical memory installed in the machine. Further, each query execution is carried out under "cold-cache" conditions. This environment is ensured by (a) restarting the database engine's server process to clean up the DBMS buffer pool, and (b) sequentially scanning a large unrelated table from the database to wipe out the operating system's cached contents, prior to executing the query.

2.1.3 Power Measurement

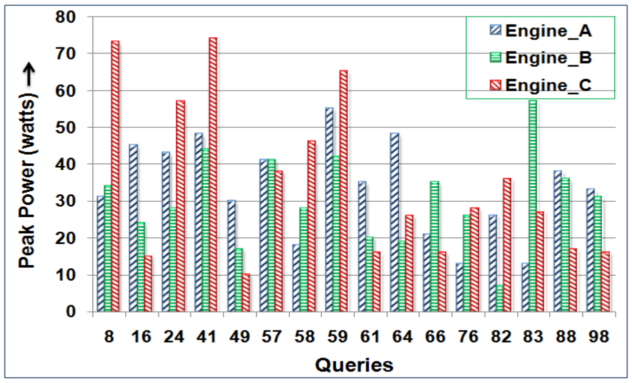
To measure power usage, we created a setup similar to those used in several prior studies [14, 15, 17, 19]. Specifically, a digital power meter (Brand Electronics model 20-1850/CI [24]) with a 1 W measurement resolution, 3 kHz sampling frequency and 1 Hz logging frequency is employed in our experiments. The meter is directly connected between the electrical mains and the database workstation, and therefore measures the workstation's overall power consumption. The power values are transmitted through an interface cable to a separate monitor machine on which they are logged and processed, thereby ensuring the measurement apparatus does not modulate the monitored system.

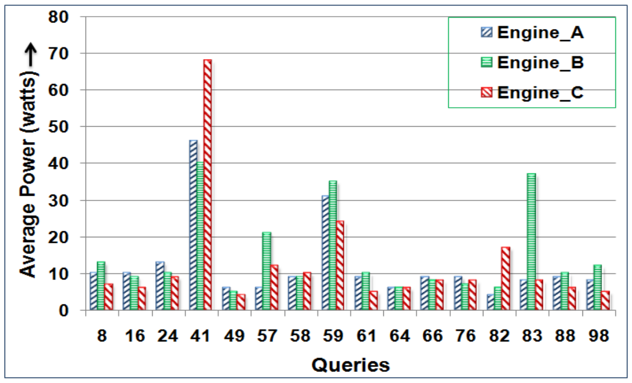
In order to obtain the active (or dynamic) power usage corresponding to database query execution, we subtracted the ambient power consumption of the system in its idle state from the measured values. For our configuration, the ambient power was around 145 W, and the saturation power value was close to 225 W, corresponding to an active range of roughly **80 W**. All measurements reported here are with respect to this range. Finally, each experiment was run with multiple independent executions to ensure confidence in the observed values.

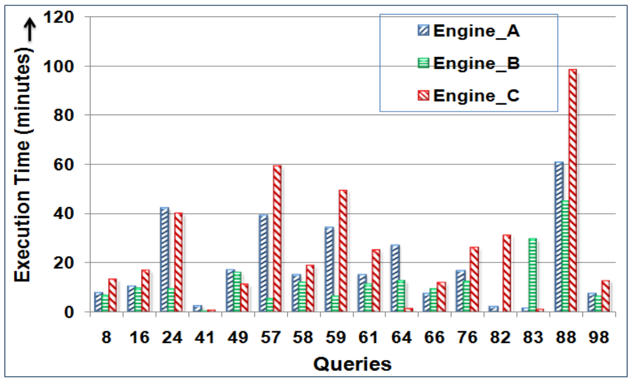
2.2 Experimental Results

Under the ambit of the above experimental framework, we evaluated the peak power values obtained on the three database engines over each of the sixteen TPC-DS queries featured in our workload. These results are presented in Figure 2(a), where EngineA is represented by blue upward diagonals, EngineB by green horizontal bars, and EngineC, by red downward diagonals. To provide the complete picture, we also show the average power values and the query execution times in Figures 2(b) and 2(c), respectively.

We first observe in Figure 2(a) that for all three engines, there exist queries spanning the various categories that exercise the underlying computational platform through a substantial range of the 80 W dynamic peak power limit. For example, with EngineA, about half-a-dozen queries (e.g. Q16) use more than 40 W, while about four do so on EngineB (e.g. Q83). Turning to EngineC, we find that it has the maximum number of power "skyscrapers", with queries such as Q8 taking in excess of 60 W. Further, there are some queries, with Q41 and Q59 being prime examples, wherein *all* three engines incur high power requirements.


(a) **Peak Power**


(b) **Average Power**


(c) **Execution Time**

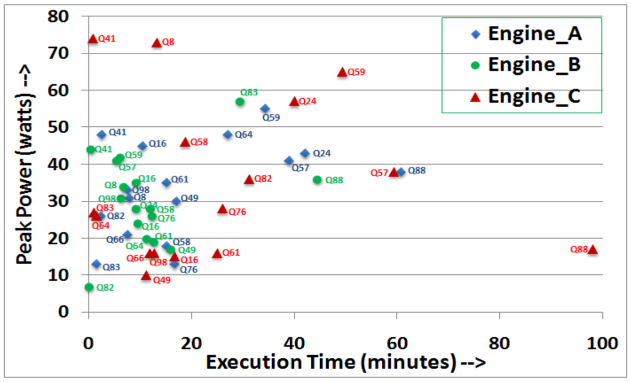

(d) **Power versus Time**

Figure 2: Power and Time Performance on TPC-DS Queries

Interestingly, with both Q8 and Q64, the average power consumption (Figure 2(b)) is roughly similar across the three engines, but their peak power behavior (Figure 2(a)) is very different. This clearly demonstrates that peak power behavior cannot be easily correlated with average power characteristics, and several such instances are present in our experiments.

The above results make it vividly evident that there is a material need to study and address the peak power consumption of database engines. We now turn our attention to juxtaposing the power and time efficiencies of the various engines. The results are shown in Figure 2(d), where the peak power is plotted against the query execution time. We observe here that EngineC displays the most extreme behavior with large execution times for some queries and high peak power values for some others. In comparison, EngineB has the best performance, mostly located near the origin, indicative of simultaneously providing good time and power efficiency. Finally, the performance of EngineA displays weaknesses similar to those of EngineC.

3 Modeling Peak Power

In this section, we move on to proposing a peak power estimator algorithm for query execution plans. While the approach itself is optimizer-agnostic, for ease of presentation we restrict our attention to modeling EngineC in the sequel. Our algorithm is based on a regression model developed from a carefully chosen set of training examples (the selection process is described in Section 3.3), with the algorithmic inputs solely based on information available in the plan descriptions provided by current optimizers. We hasten to add an important caveat here: In order to separate the estimation errors that may arise due to inaccurate optimizer estimates, as opposed to our own modeling errors, we assume for the results presented in this paper that the *correct* values for all plan parameters, such as operator input and output cardinalities, are available in the training samples (these correct values are determined through explicit execution of the sample queries). While this assumption is obviously untenable in practice, our objective here is to assess the *intrinsic* quality of our estimation model.

Average power can be easily estimated by aggregating the energy consumption estimates of each individual operator in the plan tree and dividing by the expected execution time. Peak power, on the other hand, poses the difficulty of having to account for the *concurrent* execution of a contiguous sequence of operators, commonly referred to as "pipelines". In order to identify the pipelines present in a plan tree, we leverage the prior work on SQL execution progress indicators [3, 10]. Using the algorithm presented in [3], the pipeline segmentation of the optimizer plan for benchmark query Q59 of TPC-DS is shown in Figure 3(a) – here, there are 8 pipelines, PL1 through PL8, and a partial order of the execution of these pipelines is enforced by their terminal "blocking" operators (e.g. PL4 cannot begin until PL3 is complete). Further, our analysis of EngineC suggests that it executes pipelines in an essentially serial manner, i.e. there is no inter-pipeline concurrency.

In Figure 3(a), the power figures in black rectangles on the various pipelines are the estimates from our model, and the peak power prediction for the entire execution plan is simply the *maximum* of these estimates – in this case, it would be 57.1 W. On the other hand, the power figures in red parallelograms are the actual consumption of some of the pipelines at run-time, as determined from the temporal log of the power behavior during execution, shown in Figure 3(b). (The pipeline-to-log attribution procedure is discussed later in Section 3.5.)

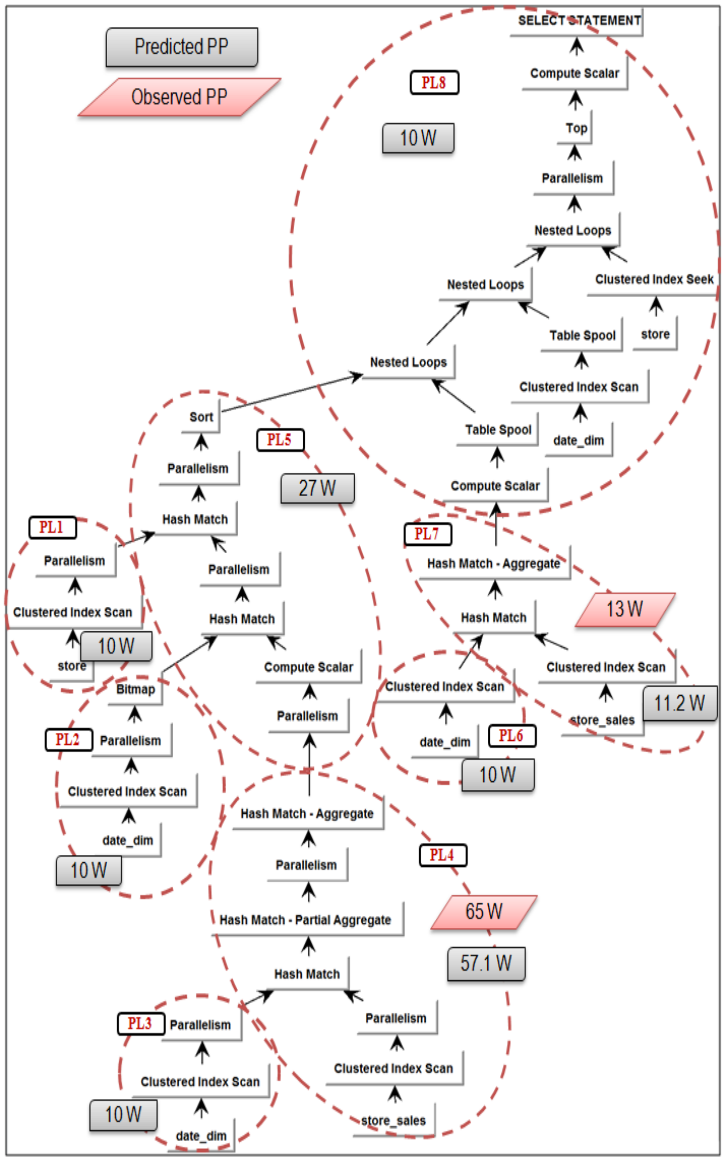
We next explain the methodology by which the peak power consumption of an individual pipeline is estimated.

3.1 Pipeline Modeling

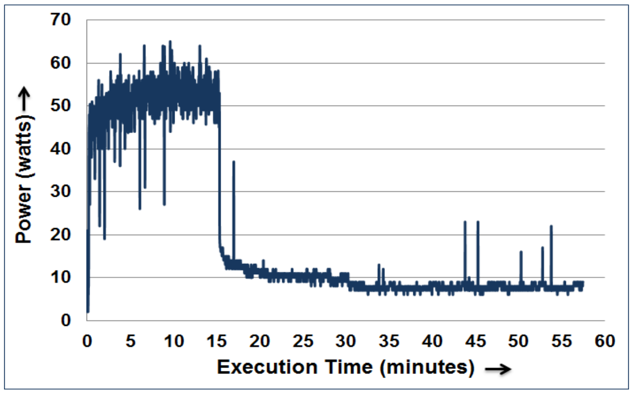
Each pipeline contains (i) a set of *driver* nodes, comprised of the operators providing inputs to the pipeline; (ii) a tree of intermediate nodes; and (iii) a single *terminal* node consisting of a *blocking* operator. (A physical operator is termed blocking if it doesn't produce any output until it has consumed at least one of its inputs completely.) An example pipeline, ePL, which features in EngineC's plans for both Q8 and Q59 (PL4), is highlighted in Figure 4. This pipeline is driven by a `Clustered Index Scan` operator and is terminated by a `Hash Aggregate` operator, with `Hash Match Join` and `Hash Partial Aggregate` being the intermediate nodes. Since the build input of the intermediate `Hash Match Join` operator is itself blocking, this input is associated with a different pipeline that has to complete before ePL's execution can commence.

In order to generate a rich diversity of pipelines, we considered queries arising not only in TPC-DS, but also in its historical precursor, the TPC-H benchmark [28]. At first glance, it might seem that we could have extended this approach even further to include other benchmarks, such as the TPC-C OLTP benchmark [26]. The reason for discounting OLTP queries is that they (a) usually do not offer a large search space for the optimizer in terms of plan alternatives, and (b) often do not run long enough to be meaningfully observed and analyzed. In contrast, the OLAP benchmarks are characterized by complex queries that execute over long time periods before reaching completion.

From the base 99 TPC-DS and 22 TPC-H benchmark queries, we created a variety of parametrized



(a) **Pipeline-segmented and Power-annotated Execution Plan**



(b) **Temporal Power Behavior**

Figure 3: Power Profile for TPC-DS Query 59

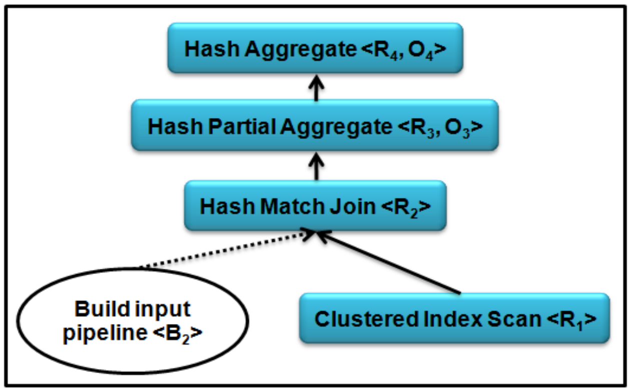

Hash Aggregate <R_4, O_4>
Hash Partial Aggregate <R_3, O_3>
Hash Match Join <R_2>
Build input pipeline <B_2>
Clustered Index Scan <R_1>

Figure 4: Example Pipeline (ePL)

	TPC-DS	**TPC-H**
Total number of Execution Plans	585	116
Total number of Pipelines	5765	711
Number of Structurally-distinct Pipelines	419	109
Number of Power-distinct pipelines	247	84
Average Length of Pipeline	8 operators	5 operators

Table 1: Pipeline Analysis of Query Execution Plans

query templates, the parameters being the predicate selectivities of a few relations appearing in the query (e.g. for Q59, the parameters were ss_sales_price $< \$1$ and d_quarter_seq $< \$2$). The values of the parametrized attributes were then varied over their domains to obtain, from the optimizer, the associated *parametric optimal set of plans* (POSP) [9] – the details of this plan generation procedure are presented in Section 5 and are also available in [16].

We then collected statistics on the kinds of pipelines present in these plans. The results are shown in Table 1, where we see that for the 585 different execution plans generated with TPC-DS, the total number of pipelines appearing in the plans is large – 5765 in all. However, most of these pipelines are structurally identical and only 419 distinct pipelines were observed. Further, when operators that are known to be power-insignificant (e.g., `Compute Scalar` and `Parallelism`) were eliminated, the number came down further to 247. For ease of presentation, we will hereafter refer to this set as the "power-distinct pipelines". In the case of TPC-H, we found 84 power-distinct pipelines from the 116 plans, with about half of these pipelines present in the TPC-DS workload as well. The average "length" of the power-distinct pipelines, measured in terms of their number of constituent operators, was 8 for TPC-DS and 5 for TPC-H, suggesting that the pipelines in the TPC-DS workload are more complex, as might be expected given the richness of its schema, data distributions and queries.

We see from the above that since the set of power-distinct pipelines is not unduly large, it should be feasible, in principle, for optimizer development teams to *a priori* model most or all of the operator pipelines appearing in their execution plans. This would facilitate integration of pipeline power models with the existing operator cost models in these systems. A related point to note here is that modeling at pipeline granularity captures both *intra-operator* parallelism and *inter-operator* parallelism.

3.2 Model Parameters

We now discuss our choice of regression model parameters. For ease of presentation, a distinction is made between two types of pipelines: (a) *Leaf Pipelines*, wherein at least one of the driver nodes to the pipeline is a leaf node in the query plan, usually corresponding to a base relation, and (b) *Internal Pipelines*, wherein all inputs are from intermediate relations, which may be hosted on disk or are fully memory-resident.

3.2.1 Leaf Pipelines

We incorporate two kinds of parameters in leaf pipelines: *Rate Parameters* and *Size Parameters*, discussed below.

Rate Parameters. Since the pipelines are fed data by driver nodes, the *rate* at which the input arrives is a critical parameter. Specifically, a scan or index operation on a disk-based relation is estimated to produce data at a rate equal to the size of the retrieved data divided by the time for transferring this data from the disk. That is, for a driver node D, the rate is computed as

$$Rate_D = \frac{Input_D}{DiskTime_D} \tag{1}$$

where $Input_D$ denotes the size of data (in bytes) retrieved from disk by D and $DiskTime$ is the disk transfer time (in seconds). While the runtime value of $Input_D$ is directly available from the API of EngineC, the same is not true for $DiskTime$ – therefore, since access to the system internals is also not feasible, the optimizer's estimate for $DiskTime$ is used instead in our study.

Now, given a pipeline PL, the rates of the downstream operators in PL are derived using the formula shown in Equation 2. Here, N is a generic downstream node in the pipeline, $Subtree_N^{\mathsf{PL}}$ is the subtree of pipeline PL rooted at node N, $Driver^{\mathsf{PL}}$ is the set of driver nodes in pipeline PL, $Source_N^{\mathsf{PL}}$ is the set of nodes in the pipeline PL that directly provide inputs to node N, and $Output_i$ denotes the size of data output by node i.

$$\begin{aligned} \text{Let } Driver_N &= Subtree_N^{\mathsf{PL}} \cap Driver^{\mathsf{PL}} \\ \text{Then } Rate_N &= \frac{\sum_{i \in Source_N^{\mathsf{PL}}} Output_i}{\max_{x \in Driver_N} DiskTime_x} \end{aligned} \tag{2}$$

The reason for the $\max$ operator in the denominator is that it selects the *slowest* driver among the pipeline's driver nodes, incorporating the assumption that the whole pipeline can only run as fast as its slowest driver. Intuitively, our approach is to model the rates of downstream nodes as the ratios of the amount of data they process to the time taken for generating the data at the head of the pipeline.

Size Parameters. In addition to data rates, we may also need to consider the sizes of the incoming and/or outgoing data for some operators in the pipeline. As a case in point, the size of the hash table for the *Hash Match Join* operator is proportional to the build input size, and therefore needs to be reflected in the model. Similarly, for the *Hash Aggregate* operator, which utilizes memory proportional to the number of output groups, the output data size is a model parameter. In our study, the runtime values for all these parameters are obtained from the API of EngineC.

Parameter	**Description**
R_1	Input rate for *Clustered Index Scan*
R_2	Input rate for *Hash Match Join*
R_3	Input rate for *Hash Partial Aggregate*
R_4	Input rate for *Hash Aggregate*
B_2	Size of build input to *Hash Match Join*
O_3	Output Size of *Hash Partial Aggregate*
O_4	Output Size of *Hash Aggregate*

Table 2: Candidate Parameters for ePL

3.2.2 Internal Pipelines

Turning our attention to internal pipelines, the driver nodes here are the blocking terminal nodes of *other* pipelines. There are two possibilities that arise:

- One or more of the driver nodes writes its data to disk, and the internal pipeline then reads this information from disk. This scenario can be treated in the same manner as leaf pipelines, using only the disk-based driver nodes in Equation 2.
- Alternatively, the outputs produced by *all* the driver nodes are small enough to be fully memory resident, resulting in the pipeline reading its *entire* input data directly from memory. This scenario is more complicated since current optimizers typically do not provide memory costs for operators. Therefore, we have taken the workaround of using purely size-based model parameters for such pipelines – specifically, the input size to each pipeline operator. Note that, as a consequence, the modeling of these pipelines needs to be carried out separately.

Parameter Example. Consider again the ePL pipeline shown in Figure 4. This is a leaf pipeline consisting of a scan, a hash join and two hash-based aggregates. For this pipeline, the associated set of candidate regression model parameters are enumerated in Table 2, and shown in the operator annotations of Figure 4. Specifically, each operator has an associated input data rate; in addition, the hash join has an input data size, while the two aggregates have output sizes, amounting to 7 parameters overall.

3.3 Generating Training Instances

Given the above modeling paradigm with the multiplicity of parameters, each covering a substantial range of values, it might appear at first glance that a computationally impractical number of training instances may be required to accurately model a pipeline's peak power behavior. However, using the methodology described next, our experience has been that even complex pipelines, running to double-digit number of operators, can be accurately modeled with a modest number of samples, typically in the range of 20 to 30. Overall, modeling the entire set of 247 power-distinct pipelines could be completed in less than three months on a single state-of-the-art workstation.

In our methodology, the first step is to decide how many samples to take. While this obviously depends on what kind of samples are subsequently chosen, an upper bound can be estimated assuming a simple random sampling of the parameter space. Specifically, given a set of desired statistical indicators (p-value, number of predictors, squared multiple correlation, and statistical power level), a sample size requirement can be calculated using the method presented in [4]. As a case in point, using standard values for the indicators, such as p-value of 5 percent and statistical power level of 80 percent, the number of suggested samples for ePL is about 40.

We now optimize on the above sample requirement by using the targeted *Latin Hypercube Sampling* (LHS) technique [12] instead of simple random sampling. The LHS approach is guaranteed to be representative of the real variability in the underlying model space, and requires that the range of each pipeline variable be partitioned into equi-probable strata, with the number of partitions being equal to the sample size. This partitioning information can be derived from the statistics and histograms that are typically available in database system catalogs.

Since it is expected that LHS will require fewer samples than random sampling [11], we *incrementally* carry out the sampling using LHS, stopping as soon as the desired statistical power for the model is reached. Using this strategy with ePL, we were able to achieve satisfactory results with only 26 samples.

Note that LHS merely indicates the desired values of the pipeline parameters in each sample. But ensuring these values is a non-trivial task since, due to our black-box environment, it is not feasible to instrument the system internals. Therefore, our mechanisms to influence the parameter values are perforce *indirect* – specifically, by varying the database schema and queries. The situation is further complicated by the dependencies existing between the various parameters (e.g. the various rates in a pipeline are correlated). Therefore the process for creating the LHS samples has to be carefully planned. An example query instance containing the ePL pipeline is shown in Figure 5. For modeling ePL, we used the following strategies to generate the training instances:

- The size of the scanned relation was altered to vary R_1. (Change location marked as R_1 in Figure 5.)

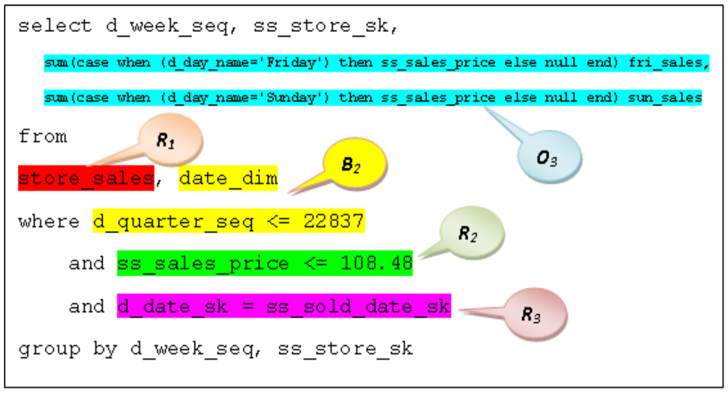
- The selectivities of the probe and build inputs were altered to vary R_2 and B_2, in the process having a follow-on impact on the values of R_3 and R_4. (Change locations marked as R_2 and B_2 in Figure 5.)

- The join conditions were altered to vary R_3 without affecting R_2 and R_1. (Change location marked as R_3 in Figure 5.)

- Various aggregates were added or modified to vary O_3, O_4 and R_4 without affecting R_3. (Change location marked as O_3 in Figure 5.)

3.4 Regression Model

Finally, to characterize peak power behavior on the training instances, we use *stepwise* multi-linear regression models. This approach is recommended when there are several candidate explanatory variables with dependencies, and no pre-defined theory on which to base the model selection [21, 22]. A beneficial side-effect is that over-fitting of the model on the training data is also reduced in the stepwise approach.



```
select d_week_seq, ss_store_sk,
   sum(case when (d_day_name='Friday') then ss_sales_price else null end) fri_sales,
   sum(case when (d_day_name='Sunday') then ss_sales_price else null end) sun_sales
from
store_sales, date_dim
where d_quarter_seq <= 22837
     and ss_sales_price <= 108.48
     and d_date_sk = ss_sold_date_sk
group by d_week_seq, ss_store_sk
```

R_1
B_2
O_3
R_2
R_3

Figure 5: A training query instance containing ePL

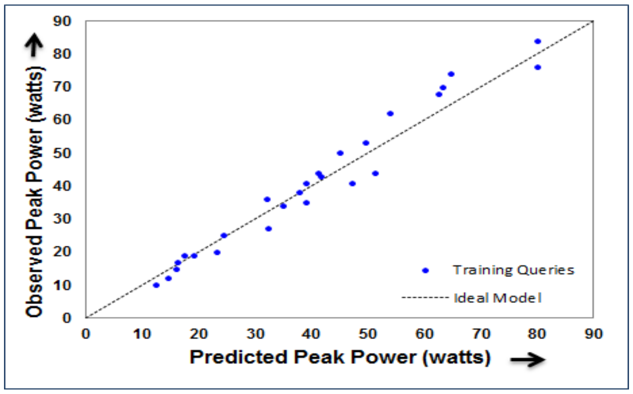


Figure 6: Regression Model on ePL

We used the XLSTAT statistical software [29] to fit the training data to a stepwise regression model. As a case in point, we optimized and executed the 26 sample queries for ePL, and from the associated query plans and executions, created the training data for all seven parameters $(R_1, R_2, R_3, R_4, B_2, O_3, O_4)$ along with the observed peak power values. The final model, shown in Equation 3, retains only *four* parameters: R_1, R_3, R_4, O_3 (which is to be expected given the underlying parameter dependencies):

$$\begin{aligned} PeakPower(\text{ePL}) &= 1.25 \times 10^{-6}\ R_1 + 7.75 \times 10^{-6}\ R_3 \\ &+ 3.67 \times 10^{-6}\ R_4 - 6.00 \times 10^{-10}\ O_3 \end{aligned} \tag{3}$$

A graph of the observed peak power values, against the fitted values from Equation 3, is drawn in Figure 6, with the dashed line signifying the ideal model. It is evident that all the training examples fall fairly close to the ideal, the overall co-efficient of variation of the RMS error being only 0.13.

Power Bounds. While the above peak power model has a reasonable fit for generic database environments, we have empirically observed on our database platform that the peak power taken by any pipeline is lower bounded by around 10W when the input rates are low, and upper bounded by 80W when the inputs are very large and the system resources are fully saturated. Therefore, we add these bounds to our peak power estimator in Equation 3.

Modeling Accuracy. As mentioned earlier, ePL features in the optimizer plans for TPC-DS queries

Q8 and Q59. The model's prediction quality on ePL in these test cases is shown in Table 3, where we see that the predicted values are in the neighborhood of the observed values.

Query Number	**Peak Power**	
	Predicted (W)	**Observed (W)**
Q8	10.7	13
Q59	57.1	65

Table 3: Modeling Quality on ePL

3.5 Results for Complete Plans

Thus far, we discussed individual pipelines. We now move on to evaluating prediction quality on *complete* query plans. Reverting our attention to Figure 3(a), corresponding to Q59, the predicted peak power value is shown for each pipeline. We intended to also measure the actual values for all these pipelines, but it proved infeasible for those that were of sub-second duration since our power meter only operates at a one-second granularity. Further, due to our black-box environment, in order to assess the peak power consumed by a pipeline, we had to *manually* look through the temporal power log and approximately identify the time segment of its execution. Owing to the complexity of the query plans, it was not always easy to make an accurate association between the temporal power log and the pipeline execution periods. However, these problems were circumvented for two pipelines PL4 and PL7, which are driven by fifteen-minute scans on the 40GB-sized STORE_SALES relation, and their observed values are shown in the red parallelograms of Figure 3(a). As can be seen, the predicted values, 57.1 W and 11.2 W, are in the ballpark of the observed values, 65 W and 13 W, respectively.

To generalize the above example, we show in Table 4 the summary set of prediction results for all the TPC-DS queries in Figure 2(a) on which EngineC consumed significant peak power – specifically, in excess of 30 W. In this table, we note that the predictions are consistently *within* $\pm$ *15 percent of the observed values*, indicating that the model is sufficiently accurate for the intended applications. Further, as mentioned earlier, if access to the engine internals were available, we expect that the accuracy could be improved even further.

TPC-DS Query	**Peak Power**		**Relative Error**
	Predicted (W)	**Observed (W)**	
Q8	74.0	72.0	+3%
Q24	53.4	58.0	-8%
Q41	78.8	74.0	+6%
Q57	34.9	38.0	-8%
Q59	57.1	65.0	-12%
Q82	38.8	36.0	+8%

Table 4: Predictions on Power-Intensive TPC-DS Queries

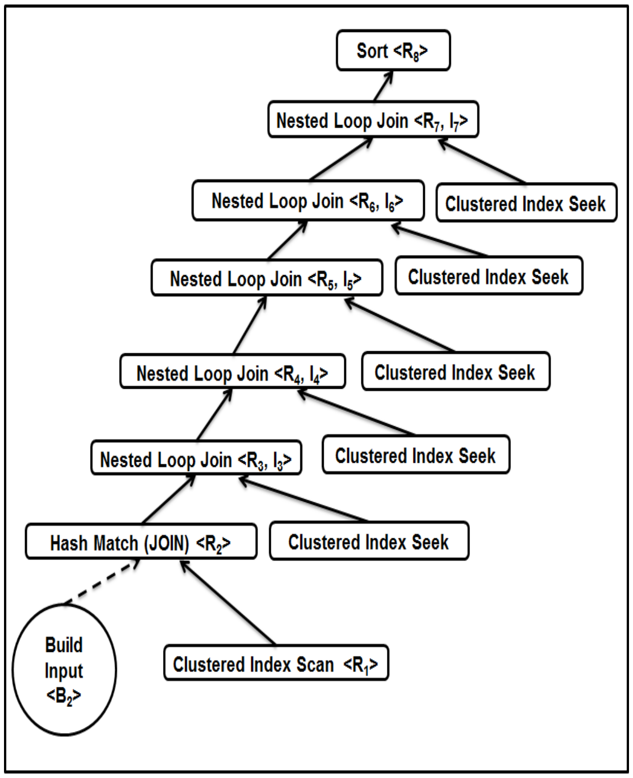

Sort <R8>
Nested Loop Join <R7, I7>
Nested Loop Join <R6, I6>
Clustered Index Seek
Nested Loop Join <R5, I5>
Clustered Index Seek
Nested Loop Join <R4, I4>
Clustered Index Seek
Nested Loop Join <R3, I3>
Clustered Index Seek
Hash Match (JOIN) <R2>
Clustered Index Seek
Build Input <B2>
Clustered Index Scan <R1>

Figure 7: Complex Pipeline cPL

4 Robustness of models

In the previous section, we illustrated how a constructive inferencing methodology for estimating peak power consumption could be set up through a pipeline-segmented modeling and training approach. We now discuss the robustness of this strategy to a variety of changes in the database system environment.

4.1 Complex Pipelines

The sizes of the pipelines in the plans considered thus far feature between 1 to 10 operators, and this range covers the vast majority of pipelines found in the TPC-DS query plans. However, we have also encountered a few instances of significantly more complex pipelines – a sample instance, cPL, consisting of 15 operators is shown in Figure 7, where an initial hash-join is followed by a sequence of five nested-loop joins. This pipeline appears in EngineC's plans for TPC-DS queries Q17 and Q25, and in our rate-and-size based modeling framework, has an associated 14 parameters.

In spite of its apparent complexity, training this pipeline did not turn out to be an arduous task. The initial estimate of sample size based on random sampling was 68, but on using LHS-based samples, a statistical power of 0.99 was achieved with only 20 instances, and the entire training was completed in around 6 hours. The final regression model obtained for cPL was the following, with just two parameters retained:

$$PeakPower(\mathsf{cPL}) = 11.1 + 1.05 \times 10^{-5}\ R_3 + 1.02 \times 10^{-6}\ I_5 \tag{4}$$

Table 5 quantitatively demonstrates that this model accurately captures the peak power consumed by cPL during the execution of the two test queries. An interesting point to note here is that the size of a pipeline does not necessarily translate to proportional peak power – cPL, for instance, only expends around 10 W. In general, our experience has been that most long pipelines consume only a modest amount of peak power. The reason can be attributed to these long pipelines typically appearing near the *root* of the plan tree. As a consequence, a significant part of the base data encountered at the leaves may have been already *filtered* before it reaches them. With small data inputs, these pipelines typically run only for short durations without consuming much resources.

Query	**Peak Power**	
Number	**Predicted (W)**	**Observed (W)**
Q17	11.12	10
Q25	11.11	11

Table 5: Modeling Quality on cPL

4.2 System Robustness

Since our pipeline-based modeling is based on inference, it may appear, at first glance to be vulnerable to changes in system configurations. To assess the impact of these changes, we attempted porting the model across different system configurations. Our experience was that changes in disk and memory configurations did not materially affect the peak power behavior; processor configuration changes, on the other hand, necessitated model retraining in some cases. To quantitatively showcase these observations, we present here the impact of each of these changes on three query plans P_1, P_2 and P_3 associated with Q59.

Effect of hard-disks. Our training server contained four 300 GB/ 15000 RPM SAS hard disks. We replaced them with four 750 GB/ 7200 RPM SAS hard disks. The peak power consumption of our test query plans remained unchanged across these two configurations, as shown by Table 6.

Effect of main memory. Here we reduced the main memory allocated to the database engine from 6 GB to 1 GB, but again, there was no appreciable effect on peak power behavior, as shown in Table 7.

Effect of degree of parallelism. Although EngineC by default uses all the processor cores present in the system, the maximum degree of intra-operator parallelism can be throttled through a configuration parameter. Specifically, on our quad-core machine, we forced it to use 1, 2 and 4 cores in three sets of experiments, respectively. Table 8 suggests that the change in the degree of parallelism (DOP) affects the peak power consumption significantly for P_1 and P_3, while P_2 remains unaffected. This is due

Query Plan	**Peak Power for 300 GB/ 15000 RPM (W)**	**Peak Power for 750 GB/ 7200 RPM (W)**
P_1	60	62
P_2	45	45
P_3	65	64

Table 6: Effect of Changing Hard-disks on Peak Power

Query Plan	**Peak Power for 6 GB (W)**	**Peak Power for 1 GB (W)**
P_1	60	61
P_2	45	44
P_3	65	63

Table 7: Effect of Changing Main Memory on Peak Power

Query Plan	**Peak Power for DOP=1 (W)**	**Peak Power for DOP=2 (W)**	**Peak Power for DOP=4 (W)**
P_1	38	50	60
P_2	45	44	45
P_3	38	48	65

Table 8: Effect of Changing Degree of Parallelism on Peak Power

to the use of a `Partial Hash Aggregate` operator by P_1 and P_3 which heavily exploits intra-operator parallelism. These experiments highlight the need for model retraining across environments with differential parallelism.

Effect of processor. Finally, we evaluated the effect on peak power consumption of a change in the base processor itelf. Specifically, we compared two systems: (i) the Intel quad-core Sun Ultra 24 workstation discussed thus far, and (ii) an AMD dual-core Sun Ultra 20 workstation.

The peak power values are again different for P_1 and P_3 for the two processors running at their default DOP, as can be seen from the first and third column of Table 9. But if we reduce the degree of parallelism in the Ultra 24 workstation from 4 to 2, the peak power values thus obtained are comparable to those obtained on the Ultra 20 workstation. This suggests that if we have a model trained for various DOPs on a multi-core processor, they can be applied to alternative systems natively sporting these configurations without requiring retraining.

Query Plan	**Peak Power for U-24 Quad core (W)**	**Peak Power for U-24 Dual core (W)**	**Peak Power for U-20 Dual core (W)**
P_1	60	50	48
P_2	45	44	46
P_3	65	48	47

Table 9: Effect of Changing CPU on Peak Power

4.3 Database Robustness

Finally, we have also evaluated the robustness of our models to large-scale changes in the database size. The important point to note here is that, unlike energy, peak power is, to the first degree of approximation, *independent* of the data size – it is affected by the data rate, not the quantity. Our experiments also indicate that pipeline peak power typically plateaus after a threshold amount of data, and our models, which were trained on 100 GB data, reflect this behavior.

We also experimentally confirmed the portability of the models built on TPC-DS to TPC-H, which is materially different in its schema, data distribution and query suite. We used a TPC-H database of size 10 GB to test the models. Table 10 shows the results on 5 test queries each containing different peak power pipelines. It can be observed that the model predictions are within $\pm 15\%$ for the new schema also, thus confirming their robustness.

TPC-H	**Peak Power**		**Relative**
Query	**Predicted (W)**	**Observed (W)**	**Error**
Test1	17.0	19.0	-10%
Test2	21.5	20.0	+8%
Test3	24.7	25.0	-1%
Test4	20.4	24.0	-15%
Test5	30.4	35.0	-13%

Table 10: Predictions on TPC-H Queries

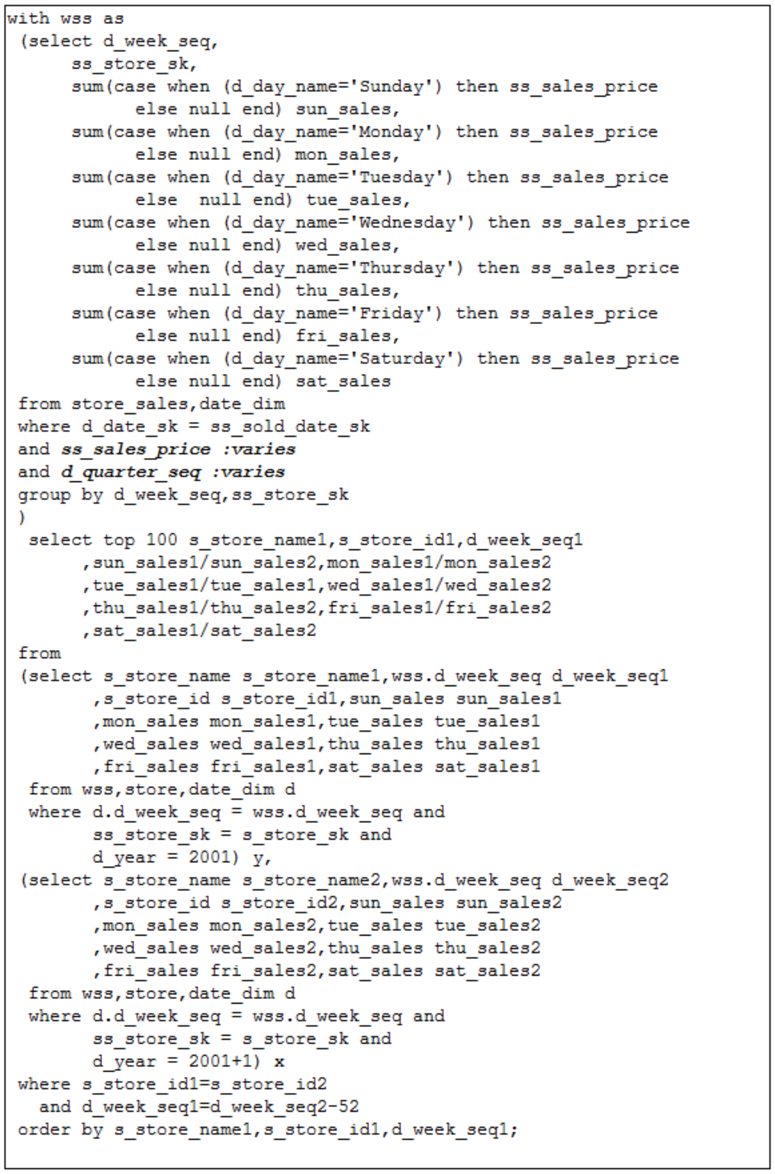
5 Power-Efficient Execution Plans

The results of the previous sections highlighted that database queries often trigger high-power bursts of energy consumption during the course of their executions. We now turn our attention to investigating how these peak power characteristics could be improved. One approach is to utilize standard power-reduction techniques such as, for example, "dynamic voltage scaling" [5]. A complementary and database-centric approach that we investigate here is to assess whether the peak power profile could be improved through a change of *query execution plans*. That is, while modern database systems typically choose the fastest executing plan, we wish to gauge whether there exist alternative plans that are more desirable from a peak-power perspective, while retaining an acceptable level of time-efficiency.

Explicitly evaluating the above approach on a database engine is predicated on the engine's support for the execution of user-specified plans, which we term as "foreign plan execution" **(FPE)**. Fortunately, EngineC, which exhibited the most extreme behavior in the experiments of Section 2, natively provides the FPE facility through its API, and we use this facility for all the results presented in this section.

Generating Alternative Plans. A related issue is the search space for alternative plans. While going through the optimizer's entire search space would provide the maximum coverage, this is obviously impractical from a computational perspective. Further, most of these plans are likely to be much worse on their time-efficiency, making them unviable alternatives. Finally, current query optimizers typically do not directly support the enumeration of alternative plans through their APIs, and it is therefore not straightforward to identify any such plan, let alone the entire search space.

To address the above issue, we take the following approach instead: We first convert the TPC-DS queries into parametrized query templates. The parametrization is on the selectivities of a subset of the base relations participating in the query, and are implemented through the incorporation of additional range predicates. This approach is also used in Section 3.1 to collect statistics on pipelines. An example query template, QT59, derived from Q59, is shown in Figure 8, where the selectivities of the STORE_SALES and DATE_DIM tables are varied through their sales_price and d_quarter_seq attributes, respectively (the associated predicates are shown in bold-face).



```
with wss as
 (select d_week_seq,
      ss_store_sk,
      sum(case when (d_day_name='Sunday') then ss_sales_price
            else null end) sun_sales,
      sum(case when (d_day_name='Monday') then ss_sales_price
            else null end) mon_sales,
      sum(case when (d_day_name='Tuesday') then ss_sales_price
            else  null end) tue_sales,
      sum(case when (d_day_name='Wednesday') then ss_sales_price
            else null end) wed_sales,
      sum(case when (d_day_name='Thursday') then ss_sales_price
            else null end) thu_sales,
      sum(case when (d_day_name='Friday') then ss_sales_price
            else null end) fri_sales,
      sum(case when (d_day_name='Saturday') then ss_sales_price
            else null end) sat_sales
 from store_sales,date_dim
 where d_date_sk = ss_sold_date_sk
 and ss_sales_price :varies
 and d_quarter_seq :varies
 group by d_week_seq,ss_store_sk
 )
  select top 100 s_store_name1,s_store_id1,d_week_seq1
       ,sun_sales1/sun_sales2,mon_sales1/mon_sales2
       ,tue_sales1/tue_sales1,wed_sales1/wed_sales2
       ,thu_sales1/thu_sales2,fri_sales1/fri_sales2
       ,sat_sales1/sat_sales2
 from
 (select s_store_name s_store_name1,wss.d_week_seq d_week_seq1
        ,s_store_id s_store_id1,sun_sales sun_sales1
        ,mon_sales mon_sales1,tue_sales tue_sales1
        ,wed_sales wed_sales1,thu_sales thu_sales1
        ,fri_sales fri_sales1,sat_sales sat_sales1
  from wss,store,date_dim d
  where d.d_week_seq = wss.d_week_seq and
        ss_store_sk = s_store_sk and
        d_year = 2001) y,
 (select s_store_name s_store_name2,wss.d_week_seq d_week_seq2
        ,s_store_id s_store_id2,sun_sales sun_sales2
        ,mon_sales mon_sales2,tue_sales tue_sales2
        ,wed_sales wed_sales2,thu_sales thu_sales2
        ,fri_sales fri_sales2,sat_sales sat_sales2
  from wss,store,date_dim d
  where d.d_week_seq = wss.d_week_seq and
        ss_store_sk = s_store_sk and
        d_year = 2001+1) x
 where s_store_id1=s_store_id2
   and d_week_seq1=d_week_seq2-52
 order by s_store_name1,s_store_id1,d_week_seq1;
```

Figure 8: Query Template 59

On this query template, we produce a "plan diagram" [16], which is a color-coded pictorial enumeration of the plan choices of the optimizer over the selectivity space defined by the template. That is, the plan diagram is a visual representation of the *parametric optimal set of plans* (POSP) [9]. As a case in point, the 2D plan diagram corresponding to the QT59 template is shown in Figure 9, drawn at a resolution of 100*100. This picture features 47 different plans, P1 through P47, with P1 (red color) occupying the largest region of the space, amounting to 24 percent.

Note that the set of POSP plans is (a) relatively very small as compared to the exponentially large search space, and (b) likely to have a reasonable time-efficiency compared to the optimizer's choice since each member is itself optimal at some region of the space.

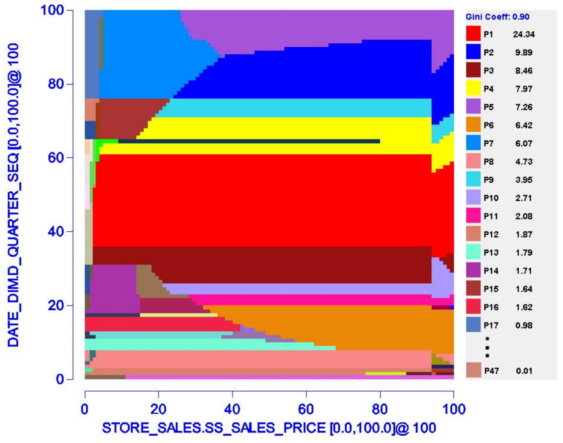


Figure 9: Plan diagram of QT59

Fortunately again, it is feasible with EngineC to provide the original TPC-DS query as input, along with any of the POSP plans corresponding to the associated template, and the optimizer automatically modifies the template plan to match the query instance. For example, when plans generated from the QT59 plan diagram are supplied to the optimizer along with query Q59, these plans are automatically modified to be consistent with the query. Using this facility, we can deterministically identify a quality set of candidate alternative plans for the query. Of course, the specific set of candidates is a function of the template that we have constructed, but a more comprehensive coverage could easily be achieved by generating a number of templates and taking the union of their POSP plan sets.

Peak-power efficient plan for Query 59. In Figure 10, we show a graph of peak power against execution time for Q59 with a suite of representative alternative plans $P_{alt1}, P_{alt2}, P_{alt3}, P_{alt4}$ from the POSP set. Observe that there is one plan: P_{alt2} (green triangle) whose peak power, 45 W, is significantly lower than the 65 W consumed by the optimizer's original plan choice P_{opt} (red square). Interestingly, in this case, P_{alt2} also happens to be *more time-efficient* than P_{opt} – however, we hasten to add that this is a serendipitous improvement arising out of weaknesses in the optimizer's cost model, and not a conscious outcome of our replacement technique.

At this juncture, the following question may be plausibly raised: Is successfully pursuing the objective of reducing peak power, predicated on incurring a substantial increase in the total *energy* consumption? We explicitly evaluated this issue for the above scenario, and found that the energy consumption of both P_{opt} and P_{alt2} is approximately the same ($\approx 70kJ$). On the other hand, if we compare the average power consumptions of the two plans, P_{alt2} is profligate by a huge margin. These results clearly highlight the fact that optimizing for average power [23], and optimizing for peak power, can result in markedly different recommendations with regard to plan choices.

The operator tree of the peak power-efficient plan P_{alt2} is shown in Figure 11(a), segmented into pipelines and annotated with predicted peak power values. The associated temporal power log is provided in Figure 11(b). From these figures, it can be seen that the model accurately predicts the peak power consumption of the two power-hungry pipelines: PL5 and PL8 (46.3 W for 45 W, 33.2 W for 35 W). It is therefore capable of correctly suggesting that P_{opt} be replaced with P_{alt2}.

Comparing P_{opt} and P_{alt2}, we find that they have significant structural differences – in particular, the initial join sequence DATE_DIM $\bowtie$ (DATE_DIM $\bowtie$ STORE_SALES) is reordered to (DATE_DIM $\bowtie$

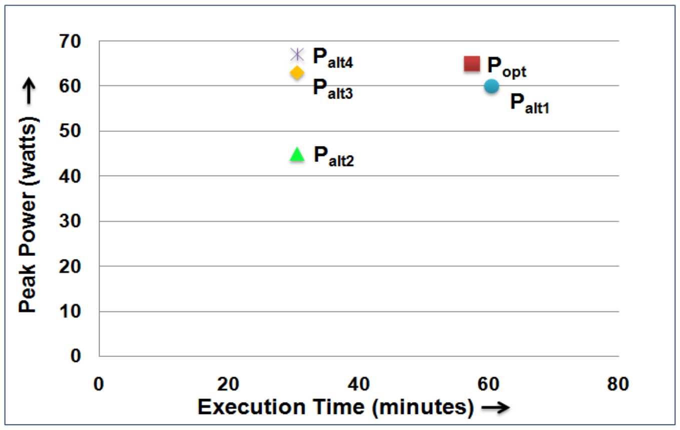


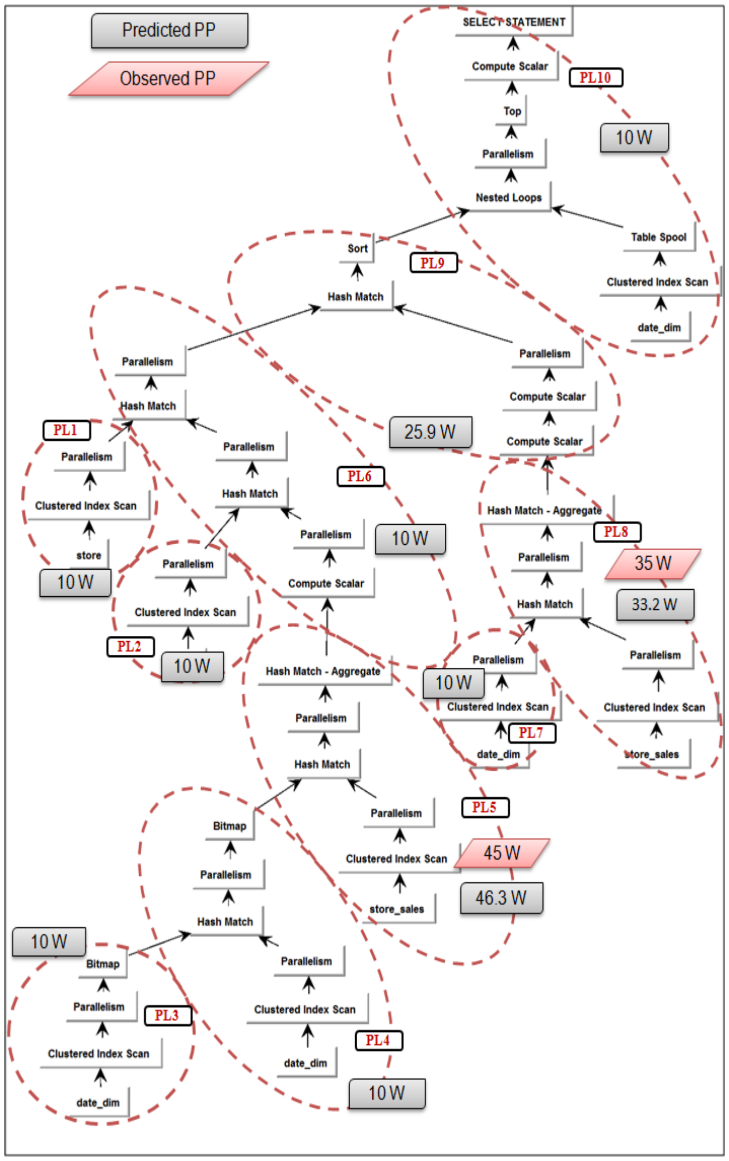
Figure 10: Peak Power against Execution Time for Query 59

DATE_DIM) ⋈ STORE_SALES, resulting in a major revamp of the pipeline structure in the plan. Specifically, the total number of pipelines increases from 8 to 10, and apart from the first 3 pipelines, no other pipeline is common between the plans.

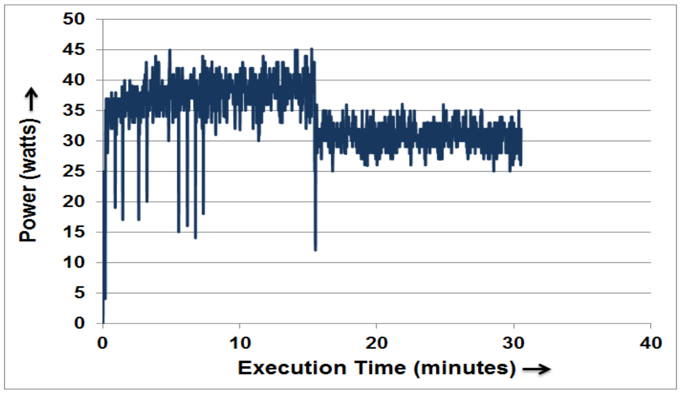
Peak-power efficient plan for Query 65. Another example query for which power-and-time efficient replacements can be identified is Query 65. This is quantitatively shown in the peak-power versus execution-time tradeoff captured in Figure 12 for a representative set of alternative plans. We see here that there are plans available, such as P_{alt2} (green triangle), which reduce the peak power consumption substantially (by about 35 W) while incurring a time penalty of around 80%. Given the conventional wisdom in the database community that a plan cost within twice (i.e. 100 percent) of the optimal is often acceptable in practice [20], it appears that P_{alt2} could be a plausible replacement choice from a holistic perspective. Finally, we also measured the increase in energy consumption, and found that it was up by around 30%, perhaps an acceptable tradeoff in light of the significant decrease in peak power.

5.1 Power Diagrams

We had introduced in [25] the notion of "plan diagrams" to represent visualizations of the plan choices made by query optimizers over an input parameter space, whose dimensions could comprise of database, query and system-related features. In analogous fashion, we introduce the notion of "power diagrams" here to represent visualizations of peak power performance over a parameter space. Consider, for example, the parameterized version of TPC-DS Q59 given in Figure 8, where the selectivities of the STORE_SALES and DATE_DIM tables are varied. A quantitative 3D power diagram showing the peak power consumption as a function of the query location in this selectivity space is presented in Figure 13. Here, the red color corresponds to the power consumption of the optimizer's time-optimal choices, whereas the green color corresponds to the best power performance at each location from among our search space of alternative plans. We notice that at some selectivity locations it is indeed possible to obtain a significant reduction of the peak power consumption. For example, at $(25\%, 75\%)$, the replacement plan reduced the peak power from 66 to 23 W, a reduction of over 40 W. Further, the number of distinct plans reduces from the original 10 to 4 in the power-efficient diagram, indicating that a few power-efficient plans can cover the vast majority of the selectivity space.



(a) **Pipeline-segmented and Power-annotated Plan**



(b) **Temporal Power Behavior**

Figure 11: Peak-power efficient Plan P_{alt2} for Query 59

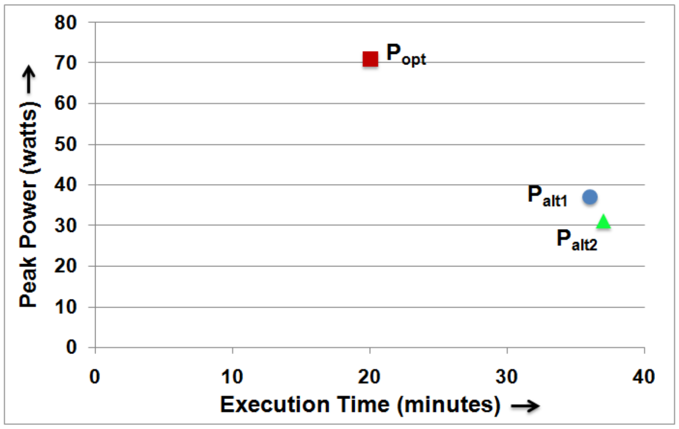


Figure 12: Peak Power versus Execution Time for Query 65

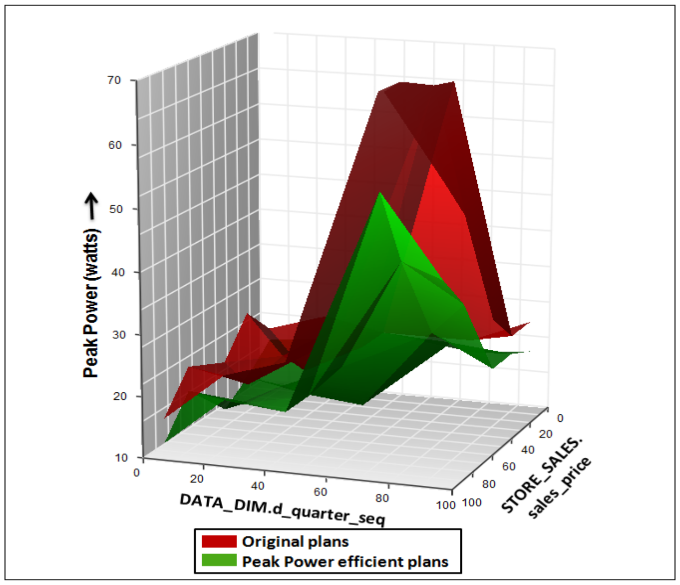


Figure 13: Peak power diagram for QT59

Note that these improvements are *conservative* since the search space is POSP-limited – if access to the complete search space were available, plans with even better power profiles may be identified.

6 Modeling Extensions

In the previous sections, we have presented the basic mechanisms for profiling and utilizing peak power behavior. We now discuss a variety of ways in which this framework could be extended to enhance these capabilities.

6.1 Inductive Modeling

A new pipeline may often turn out to be an extension of a previously modeled pipeline. For example, we may encounter a pipeline with $n + 1$ hash-joins after having previously modeled the n hash-joins scenario. In this situation, it would be beneficial if the existing model could be *incrementally* extended to handle the additional join operator. A preliminary assessment of this issue has yielded promising results for hash-join sequences, as explained next.

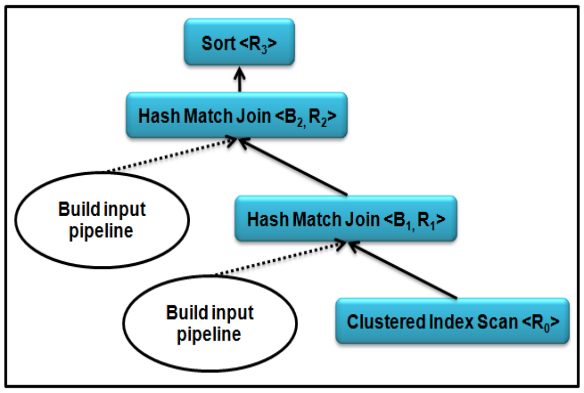


Figure 14: Base Pipeline with 2 Hash-Joins

We initially analyzed a pipeline with a sequence of two hash joins terminated by a sort operation (shown in Figure 14), coming up with the following model, where B_2 is build input size of second hash join and R_2 is its input data rate:

$$PeakPower_2 = 23.7 + 1.8 \times 10^{-6}\ B_2 + 4.0 \times 10^{-5}\ R_2 \tag{5}$$

This model was then generalized to the case of $n+1$ $(n \geq 2)$ hash-joins through the *recurrence* shown in Equation 6 (using PP as shorthand for PeakPower):

$$PP_{n+1} = K \times PP_n + \alpha_{n+1} + \beta_{n+1} \times B_{n+1} + \gamma_{n+1} \times R_{n+2} \tag{6}$$

where B_{n+1} denotes the build input size of the additional hash join, R_{n+2} denotes the output data rate of the hash join, identically equivalent to the input data rate of the blocking sort operator, α_{n+1}, β_{n+1} and γ_{n+1} being the associated parameter coefficients; PP_n denotes the peak power of the same pipeline with n hash joins; and K reflects the "back-pressure" impact of the additional join on the upstream operators. Note that the number of training instances constructed for these inductive equations are reduced as compared to those required for the corresponding native "developed-from-scratch" model since now the equation is predefined and only the values of the coefficients have to be identified. As a case in point, the number of training samples required for all the inductive hash-join pipelines comes down to 11, as compared to the 16 used by the native models for comparable accuracy.

With this approach, the following equations were developed for pipelines containing 3, 4 and 5 hash-joins, respectively:

$$PP_3 = -71.2 + 0.05\, PP_2 + 5.7 \times 10^{-6}\, B_3 \tag{7}$$
$$PP_4 = -46.2 + 0.33\, PP_3 + 5.4 \times 10^{-6}\, B_4 \tag{8}$$
$$PP_5 = 7.65 + 0.49\, PP_4 + 1.3 \times 10^{-7}\, B_5 \tag{9}$$

The prediction quality of each of these recurrence-based models is shown in Table 11, for a variety of test-cases. We observe from the results that the relative error is always within $\pm$ 15%. Further, for reference purposes, the accuracy of the associated native model is also given in Table 11, and we see that the two predictive models provide comparable performance.

Test Query	Inductive Prediction (W)	Observed Peak Power (W)	Native Prediction (W)
3 Hash Join Pipeline			
Test1	48	42	34.8
Test2	45.3	47	42.6
Test3	12.6	12	11.8
4 Hash Join Pipeline			
Test4	53	50	46.1
Test5	13	10	13
5 Hash Join Pipeline			
Test6	57	61	53.7
Test7	20	18	17.3
Test8	40	47	43.2

Table 11: Inductive Modeling Accuracy

Compact Inductive Models

We can leverage the ideas of the previous section one step further by taking a closer look at the recurrence, and realizing that the build size of the new hash join and its output data rate are correlated. Therefore, the data rate alone may prove sufficient to express the behavior of the new join. With this observation, we modified the recurrence to that shown in Equation 10,

$$PP_{n+1} = K \times PP_n + C_{n+1} + A_{n+1} \times R_{n+2} \tag{10}$$

where R_{n+2} denotes the output data rate of the additional hash join, C_{n+1} and A_{n+1} being the associated parameter coefficients; other terms remaining the same. With the new expression, the number of training instances came down to just 7, as compared to the 11 used by the full-blown inductive model.

The following new equations were developed for pipelines containing 3, 4 and 5 hash-joins, respectively:

$$PP_3 = -26.2 + 0.71\, PP_2 + 8.3 \times 10^{-4}\, R_4 \tag{11}$$
$$PP_4 = 2.3 + 0.71\, PP_3 + 9.3 \times 10^{-4}\, R_5 \tag{12}$$
$$PP_5 = 26.7 + 0.71\, PP_4 + 2.0 \times 10^{-4}\, R_6 \tag{13}$$

These “compact” inductive models maintain comparable accuracy to the full-blown versions as can be observed from the results in Table 12, where their performance is characterized over the same suite of test queries used in Table 11.

6.2 Multi-query Workloads

So far, we considered the TPC-DS queries to be executing one at a time, in isolation. In practice, however, there may be multiple queries that are concurrently executing and exercising the system resources. Therefore, an interesting research problem is to investigate how the single-query models could be extended to accurately capture multi-query environments, based on which database administrators could employ admission-control or load-control strategies to ensure that the desired peak-power threshold is not breached.

Test Query	Inductive Prediction (W)	Observed Peak Power (W)
3 Hash Join Pipeline		
Test1	36	42
Test2	53	47
Test3	10	12
4 Hash Join Pipeline		
Test4	49.7	50
Test5	10	10
5 Hash Join Pipeline		
Test6	60.8	61
Test7	16.3	18
Test8	40	47

Table 12: Accuracy of Compact Inductive Models

This policy needs an estimate of the peak power for each constituent query and also the *location* of the peak during its execution timeline. Our peak power model is capable of identifying the pipeline drawing the maximum peak power in a given query. If we integrate this knowledge with the time (cost) estimates given by the optimizers, we can roughly identify the location of the peak power consumption in the query's timeline. This model can then be used for appropriately scheduling the queries.

To assess the above, we carried out exploratory experiments with two concurrent queries. A few results for the case when the query pairs are "data-disjoint", that is, they do not share any inputs, are presented in Table 13. These results indicate that the peak power of the combined workload can be approximated by merely taking the maximum of the independent peak powers of the two queries. This matches with our expectation since only a single pipeline is in execution at any given time in EngineC, independent of the number of concurrent queries.

Workload Queries	Peak Power of First Query (W)	Peak Power of Second Query (W)	Peak Power of Workload (W)
(Q8, Q24)	72	58	76
(Q58, Q59)	46	63	55
(Q41, Q58)	74	58	75
(Q41, Q82)	74	36	78
(Q24, Q57)	58	38	56

Table 13: Observed Peak Powers for Multi-query Workloads with No Data Sharing

On the other hand, if the two queries happen to *share* a portion of their inputs, then some leaf pipelines may behave like memory-resident internal pipelines due to one query bringing into memory the inputs required by the other. In this scenario, it is hard to know in advance the temporal sequencing between interacting pipelines, especially if they commence at staggered time instants. Since we are limited to being outside the database engine, instead of trying to predict the peak power itself, we tried to establish empirical *bounds* on its values, corresponding to best-case and worst-case scenarios, respectively. In the best-case, when no pipeline is memory-resident, the peak power predictions can be made using the maximum-power heuristic given above, and this value can be used as the lower bound. On the other hand, the worst case corresponds to when all input-sharing pipelines are modeled as memory-resident internal pipelines instead of leaf pipelines, and the peak power prediction for this scenario is used as the upper bound.

Our experiments indicate that the actual consumption for various temporally random combinations of the query pair, always fell within the estimated range. In particular, to explicitly simulate the worst case, we evaluated query workloads wherein two instances of the *same query* were started simultaneously and run concurrently. As a case in point, when the workload consisted of two concurrent instances of Q59, the observed peak power was 70 W, while the range predicted by our model is [57 W, 70 W]. A more detailed analysis of how the range could be interpolated to make accurate predictions for specific data-sharing regimes is left for future work.

6.3 Guidelines

In closing, we highlight a few observations that may be of benefit to database system developers and administrators with regard to tuning their systems for meeting peak power budgets.

We have found that CPU-intensive operations typically draw more instantaneous power than disk-based operations – for example, pipelines involving CPU-intensive operators such as `Hash Match Join` and `Aggregate` are found to draw high peak powers, whereas the pipelines dominated by operators such as `Nested Loops Join` and `Clustered Index Scan` consume lower peak power. Therefore, whenever the associated response-time penalties are acceptable, a simple heuristic of avoiding hash-based operators may be employed to lower the peak power consumption.

As mentioned earlier in this section, our models are also useful in capping peak power for multi-query workloads. This is feasible because our model, based on serial execution of pipelines, is able to predict not only the peak power incurred by each constituent query but also estimate the location of the peak power burst during the workload's overall timeline. Database administrators can exploit this information to re-schedule the queries to maintain the peak power under a threshold value.

Finally, our experience has been that most long pipelines consume only a modest amount of peak power – that is, short pipelines tend to be the peak power culprits in query plans. This is perhaps due to long pipelines typically appearing near the *root* of the plan tree – consequently, a significant part of the base data encountered at the leaves may have been already *filtered* before it reaches these pipelines. With small data inputs that are consumed rapidly, it is not feasible to sustain the data rates required to drive these pipelines upto their maximum power consumption, and therefore their effects are are not prominent in the power log. From a training perspective, this suggests focusing attention on short pipelines if the training budget is limited.

7 Related Work

During the last few years, the redesign of database engines to gain efficiency on energy-related issues has been increasingly viewed as a promising approach. For example, software developers are challenged in [7] to develop energy-efficient databases through reworking optimization choices, scheduling algorithms, physical database designs and database update techniques. These thoughts are echoed in the insightful views of [8] wherein experimental evidence is provided to demonstrate that current query optimizers may not choose energy-efficient plans. Energy-aware enhancements through leveraging system-wide tuning knobs and query optimizer parameters are suggested, and the need for rethinking database algorithms and policies is emphatically made.

There has also been work on modifying the database query optimizer to choose more energy efficient query plans. Interestingly, the first such attempt was in [2], almost two decades ago. Here, the goal was to increase the effective battery life of mobile computers by selecting energy-efficient query plans, using a energy predictor model developed from optimizer cost estimates and system parameters. Since a client-server framework was assumed, their emphasis was on optimizing the network throughput and overall energy consumption. More recently, plan-based energy management schemes for memory-resident databases on banked memory architectures were proposed in [13]. Here, query execution plans are explicitly augmented with turn on/off instructions for individual memory banks, and these plans are then restructed and regrouped to gain energy efficiency. A simulation-based study of the scheme provided promising results but these observations are yet to be validated on real systems.

The study of average power behavior in database query optimizers presented recently in [23], is perhaps the closest to our current work. Here, opportunities for power savings in current database optimizers are initially highlighted. Then, the query optimizer is modified to take power costs explicitly into account with an average consumption power model developed on the lines of PostgreSQL's cost model. Their results indicate that it is possible to identify execution plans with attractive tradeoffs between average-power and time-efficiency.

In a concurrent research study, a thorough investigation of both hardware and software knobs to improve energy efficiency on PostgreSQL and a commercial database engine, was presented in [19]. Since they evaluated the power consumed by the *complete system*, which was significantly larger than the dynamic consumption incurred solely through query processing, their experimental results suggested that going with the time-optimal configuration or plan usually resulted in the best power efficiency also. However, given the strong ongoing efforts by the hardware and OS communities to reduce the idle power consumption, it is likely that we will soon encounter situations wherein the dynamic power consumed by query processing can become a significant factor, and our work is predicated on this eventuality. The potential for using software mechanisms to cap peak power consumption of database systems was also highlighted in [19], and our study attempts to quantitatively substantiate these views on industrial-strength platforms.

In a parallel effort to ours, modeling of peak power using regression techniques was also recently attempted in [18]. However, their model has been evaluated only with simple selection queries on single relations and therefore does not reflect the effect of pipelines or rate-based parameters. In contrast, our work has been carried out in industrial-strength benchmark environments.

8 Conclusions

We have investigated here, for the first time, the peak power behavior of modern database engines while processing complex SQL queries. Our "black box" study of a representative set of commercial database engines on the TPC-DS benchmark shows that the peak power consumption could be quite significant, covering the entire dynamic range of the underlying computing platform, which in our case was 80 watts. The results also bear testimony that the peak power behavior could be considerably different to the corresponding average power behavior, highlighting the need for studying these metrics separately.

We proposed a pipeline-based model for predicting the peak power consumed by query execution plans, developed through step-wise linear regression over training instances that were carefully chosen using the targeted and efficient LHS sampling scheme. Since access to the system internals was not available, these instances were indirectly created through variations in the database schema and queries. Our initial experimental results indicate that the model, which only uses generic plan-based parameters as inputs, is reasonably accurate in its predictions, with an error of less than 15%. Further, we also indicated how pipeline modeling could be inductively carried out as an extension of prior models, incurring far less overheads as compared to ab initio development.

We also demonstrated that while current optimizers typically choose the most time-efficient plan, often alternatives exist that are significantly more peak-power-efficient without unduly compromising the query running time and the overall energy. Further, the notion of "power diagrams" was introduced and it was shown that power-efficient plans covering large selectivity spaces could be identified from the POSP set. These observations serve to encourage the design of query optimizers that organically include power characteristics as a selection metric during their exploration of the plan space. In our future work, we intend to integrate and implement these ideas within the PostgreSQL engine.

9 Acknowledgements

This work was supported in part by grants from Microsoft Research and Dept. of Science & Technology, Govt. of India. We thank Prof. Matthew Jacob for his advice on the experimental methodology and the LHS sampling approach. We also gratefully acknowledge the assistance of Mahesh Bale and Bhavana Ganji in the implementation and experiments.

References

[1] R. Agrawal et al, "The Claremont report on database research", *SIGMOD Record*, 37(3), 2008.

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

[3] S. Chaudhuri, V. Narasayya and R. Ramamurthy, "Estimating progress of execution for SQL queries", *Proc. of SIGMOD* 2004.

[4] J. Cohen, "Statistical power analysis for behavioral sciences (2nd edition)", *Lawrence Earlbaum Associates* 1988.

[5] E. Elnozahy, M. Kistler and R. Rajamony, "Energy-efficient server clusters", *Proc. of PACS* 2002.

[6] W. Felter, K. Rajamani, T. Keller and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems", *Proc. of ICS* 2005.

[7] G. Graefe, "Database servers tailored to improve energy efficiency", *Proc. of SETMDM* 2008.

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

[9] A. Hulgeri and S. Sudarshan, "Parametric query optimization for linear and piecewise linear cost functions", *Proc. of VLDB* 2002.

[10] G. Luo, J. Naughton, C. Ellmann and M. Watzke, "Toward a progress indicator for database queries", *Proc. of SIGMOD Conf.*, 2004.

[11] A. Matala, "Sample Size Requirement for Monte Carlo simulations using Latin Hypercube Sampling", Mat-2.4108, Helsinki Univ. of Techology, 2008.

[12] M. McKay, R. Beckman and W. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code", *Technometrics* 1979.

[13] J. Pisharath, A. Choudhary and M. Kandemir, "Reducing energy consumption of queries in memory-resident database systems", *Proc. of CASES* 2004.

[14] M. Poess and R. Nambiar, "Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results", *PVLDB*, 1(2), 2008.

[15] M. Poess, R. Nambiar and D. Walrath, "Why you should run TPC-DS: a workload analysis", *Proc. of VLDB Conf.*, 2007.

[16] N. Reddy and J. Haritsa, "Analyzing plan diagrams of database query optimizers", *Proc. of VLDB Conf.*, 2005.

[17] S. Rivoire, M. Shah, P. Ranganathan and C. Kozyrakis, "JouleSort: a balanced energy-efficiency benchmark", *Proc. of SIGMOD Conf.*, 2007.

[18] M. Rodriguez-Martinez, H. Valdivia, J. Seguel and M. Greer, "Estimating Power/Energy Consumption in Database Servers", *Procedia Computer Science*, Vol. 6, 2011.

[19] D. Tsirogiannis, S. Harizopoulos and M. Shah, "Analyzing the energy efficiency of a database server", *Proc. of SIGMOD Conf.*, 2010.

[20] F. Waas and C. Galindo-Legaria, "Counting, enumerating, and sampling of execution plans in a cost-based query optimizer", *Proc. of SIGMOD Conf.*, 2000.

[21] L. Wasserman, "All of Statistics", *Springer* 2004.

[22] S. Weisberg, "Applied Linear Regression", *Wiley* 1985.

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

[24] "Brand Electronics Power Meters". `http://www.brandelectronics.com/meters.html`

[25] "Picasso Database Query Optimizer Visualizer". `http://dsl.serc.iisc.ernet.in/projects/PICASSO/`

[26] "TPC-C Transaction Processing Benchmark". `http://www.tpc.org/tpcc`

[27] "TPC-DS Decision Support Benchmark". `http://www.tpc.org/tpcds`

[28] "TPC-H Decision Support Benchmark". `http://www.tpc.org/tpch`

[29] "XLSTAT Statistical Analysis Software". `http://www.xlstat.com/`