PEAK POWER ANALYSIS AND MODELLING

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Engineering

IN THE

COMPUTER SCIENCE AND ENGINEERING

by

PUNEET KUMAR BIRWA

Department of Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

July 2011

TO

My Family

Acknowledgements

First of all I would like to thank my Prof. Jayant R. Haritsa for giving me the opportunity to pursue my project under his guidance. I thank him, for his constant and valuable guidance, support and encouragement during my stay at IISc.
I would like to thank my friends at IISc. who have made my stay at IISc memorable and helped me out at last moments.
I thanks my family for their continued support throughout my career.

Abstract

Database engines often consume significant power during query processing activities especially during complex query processing, which is motivating researchers to investigate the redesign of their database internals to minimize the energy 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 environments, and demonstrate that their consumption can often be substantial and we have also shown that average power consumption behavior of optimizers are very different from the peak power consumption behavior of queries. 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, the model can be incorporated in current query optimizers to identify relatively "green" plans. We demonstrate sample instances of this application wherein power-hungry plans are replaced with alternatives that substantially reduce peak power requirements, without materially compromising query execution times. In the end we also show our preliminary work on inductive pipeline (i.e. when we have modeled a pipeline and a new pipeline comes which has just one join operation more so we can predict its peak power) and multiquery environment (when more than one queries are running over a single server.

Contents

List of Tables

List of Figures

Keywords

Database, Peak Power, Regression, Pipeline, Operators, Rate, Input and Output Size, Dynamic Power.

Chapter 1

Introduction

Technological advances, environmental concerns and mobility considerations has fuelled the power consumption of computational hardware and software, because of which addressing the power consumption of it has become an active area of research. As Database engines are one of the key component of many enterprise information systems and it has been seen that they are major power consumers during the data processing activities specially during the complex query execution. The above fact led the Claremont report on database research directions [2] to highlight *"designing power-aware DBMSs that limit energy costs without sacrificing scalability"* as an important research area.

Concerns of Power. There are two aspects of power that are to be considered during the evalution of power utilization. First is *Average power* whose impacts concerns such as long-term energy expenses and design of heat dissipation systems, whereas the second is *peak power* which is of relevance in server design, capacity planning, and prevention of overheating surges. Also for peak power, it is mentioned in [9] that *since cooling and power supplies are designed to accomodate peak consumption, reducing this overhead mitigates power and cooling limitations*

Studies over Average power consumption in database engines were already done in prior literatures (covered in detail in Chapter 6). But recently, the challenge of developing database software that tunes the query execution to meet a given power budget was posed in [24]. Accordingly, we turn our attention here to profiling and modeling the *peak power* characteristics of database engines. New problems are posed here since we have to now explicitly account for :

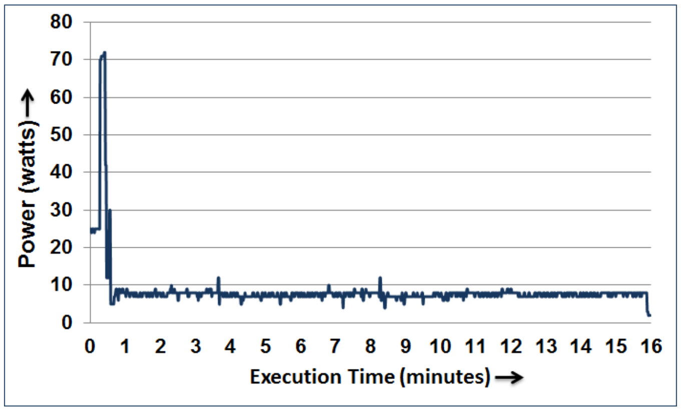
1. The *parallelism* of operators, as peak power represents the maximum aggregate consumption of concurrent operations; and
2. Capture what could well turn out to be *bursty* or short-term phenomena during the course of a query's execution.

Our work begins with the profiling of 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 benchmark [33]. After this we try to model the peak power of a query plans.

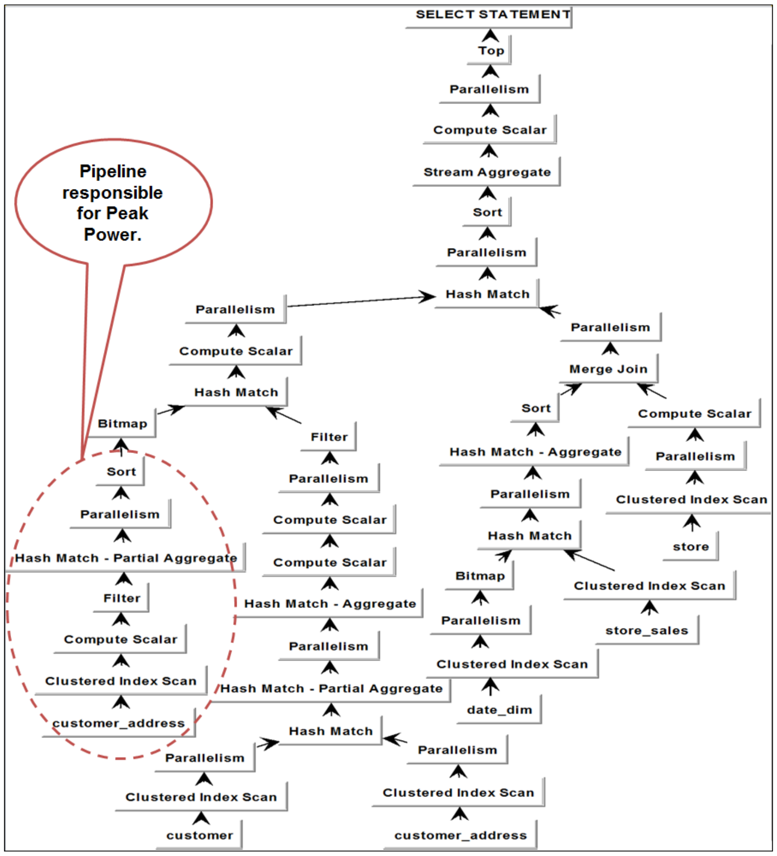
TPC-DS benchmark It models an industrial-strength data warehousing environment with complex decision-support queries and large schema. It is considered to be an excellent platform for testing query processing systems [20].

On performing various experiments it was inferred that, even when the queries are executed in isolation the power consumption incurred by such query processing can often take up a substantial fraction of the machine's dynamic power range. Often there are differences in the peak power consumption of the various engines. As an example , 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 1.1(a), which tracks the power consumption over the query's 16 minute lifetime. And, its source can be traced back to the "pipeline" (sequence of concurrently executing operators) segment highlighted in the execution plan tree shown in Figure 1.1(b).

Regression Model. Above obtained empirical observations motivated us to investigated about the possibility to *a priori* estimate the peak power consumption of a query. Our interest was to know if this estimation could be carried out solely using information provided by the query execution plan, without requiring any run-time inputs.



(a) **Temporal Power Behavior**



(b) **Optimizer Plan**

Figure 1.1: TPC-DS Query 8

The main concern is that on today's multi-core computing platforms, multiple operators may be executing in parallel, 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 funnelling data into the pipeline. The next step was to develope a model based on these observations, wherein a query plan is first segmented into pipelines, using techniques developed previously for resource management in parallel query processing [6] and for SQL progress indicators [5, 14]. A mathematical function has been developed through fitting step-wise linear regression models [25, 26] on a small set of training examples. This function that takes as input the data rates and sizes of the inputs and outputs of the pipeline operators, is applied on each of these pipelines. This gives an estimate of the peak power consumption. Our evaluation shows that this model is typically able to estimate peak power within 15% of the consumption encountered at run-time, when the plan parameters are accurately modeled in the database system. Therefore, it appears to be a useful tool for incorporation in the design workbench of database servers.

Optimizer Integration. As per our knowledge, the selection of query execution plans by modern database engines is based on minimizing the estimated query execution time, power considerations are currently not directly taken into account. In this scenario, it is entirely possible that 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 power-efficiency of the various plan alternatives considered by the optimizer, thereby supporting making weighted choices between power and time considerations. Following steps were followed to explicitly assess this possibility:

- Sample TPC-DS queries were converted into parametrized templates.
- Then, by invoking the query optimizer at various locations over the parameter space, we produced a *parametric optimal set of plans* (POSP) [12], over this space.
- Next, using our model, we evaluated the feasibility of replacing power-hungry plans

chosen by the optimizer with POSP alternatives that materially reduce peak power requirements without incurring excessive increases in query execution times.

As a specific example, we were able to reduce the peak power by around *20 to 40 watts* in some instances, a significant reduction given that the dynamic power range of our testbed machine was of the order of 80 watts. Further, these improvements were obtained at a relatively modest execution-time increase. In fact, in some cases, we were serendipitously able to even achieve *improvements* in execution-time, due to inaccuracies in the optimizer's cost model pushing forth sub-optimal plan choices.

Black Box Environment. Our entire study has treated the commercial database systems as "black boxes", utilizing only the API functions provided by the query engines due to not being privy to the internals. Thus our attribution of plan operator activity to the temporal power behavior in the training examples is perforce a coarse association. We could improve the accuracy in the peak power estimation functions if the correspondence could be established more precisely. Which in turn depends on the availability to access the engine internals. Further, it would be feasible to consider power-efficient replacements from the enormously larger native *plan search space*, rather than merely the restricted POSP space constructed by us.

In this thesis we attempt to demonstrate a proof-of-concept that it is indeed feasible to efficiently extend current database engines to be peak-power-conscious, taking another step towards the ultimate objective of designing "green" database systems. To the best of our knowledge, our results represent the first peak power characterization of database query processing.

Organization Of Thesis. The remainder of this thesis is organized as follows: In Chapter 2, we profile the peak power performance of a representative set 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 Chapter 3. Then, in Chapter 4, 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 extensions to the work described here are discussed in Chapter 5. The prior literature on profiling and improving power characteristics of database systems is reviewed in Chapter 6. Finally, in Chapter 7, we summarize our conclusions.

Chapter 2

Peak Power Profiles

This chapter contains the results of profile the peak power behavior on a representative set of three popular commercial relational database engines. These engines are anonymously referred to as *EngineA* , *EngineB* and *EngineC* in the sequel.The workload consists of complex SQL queries sourced from the TPC-DS decision support benchmark [33]. Now we explain the enviornment in which our experiments were done.

2.1 Experimental Environment

Our experiments were conducted on system platform with the specifications mentioned in Table 2.1.

We have used a Scale 1 version of the TPC-DS benchmark, corresponding to a disk occupancy

Platform	sun Ultra 24 workstation
Processor	Intel Core 2 Extreme Quad Core 3GHz
RAM	8 GB
SAS hard disks	four 300 GB (15K RPM)
Operating system	64-bit Windows Vista Business

Table 2.1: System Specification

of about 100 GB was used to populate the database for our experiment. [1]

2.1.1 Query Workload

Results are presented based on an illustrative subset of 16 queries from the 99 SQL queries that feature in the TPC-DS benchmark. Classification of queries based on their coverage of fact and dimension tables in the data warehouse schema is present in [20]. The choice of these queries was motivated by this categorization. The classification is as follows:

- Queries involving dimension tables only (6 queries)
- Queries involving a single fact table (54 queries)
- Queries involving multiple fact tables with join of sub-queries (22 queries)
- Queries involving multiple fact tables with union of sub-queries (17 queries)

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

The assigned memory is set to the same value for each database engine, namely 6 GB of the 8 GB physical memory installed in the machine. Further, each query execution is carried out under “cold-cache” conditions. “cold-cache” is the conditions wherein the data required for the execution of the query is not present in the memory. Steps followed to ensure this environment are:

- Restarting the database engine’s server process to clean up the DBMS buffer pool, and

[1] EngineA , EngineB *and* EngineC *corresponds to* the three database engines IBM DB2 9.7, Oracle 11g and Microsoft SQL Server 2008 respectively.

- Sequentially scanning a large unrelated table from the database to wipe out the operating system's cached contents, prior to executing the query.

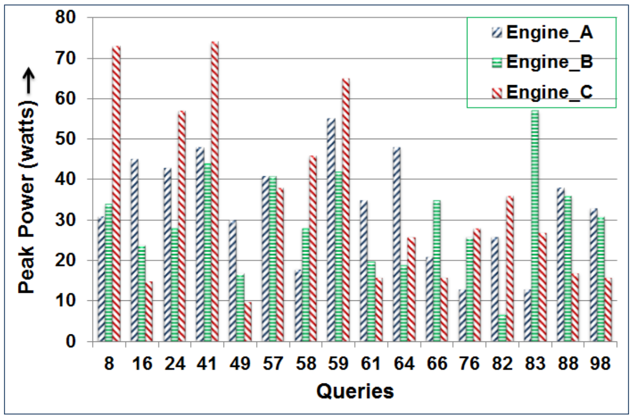
2.1.3 Power Measurement

To measure power usage, a digital power meter (Brand Electronics model 20-1850/CI [30]) with a resolution of 1 Watt and a sampling frequency of 1 Hz, was 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. This is the same measurement setup used in [19, 20, 22, 24]. Ideally, we would like to separately measure the consumption at various hardware resources such as the processor, memory and hard disks. However, since this required considerably more instrumentation, we chose to use a single hardware meter in conjunction with the Windows built-in resource activity meters to divvy up the cumulative usage across the various components. The power meter has a computer interface cable through which the measured power values are transmitted, and these were logged and processed on a separate monitor machine to ensure that they did not influence the measurements.

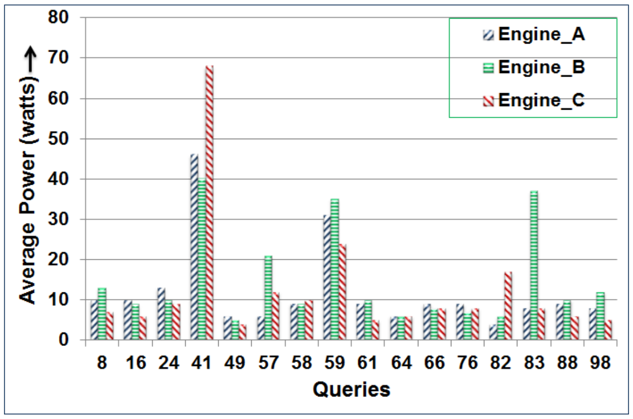
In order to obtain the active or dynamic power usage corresponding to a database query execution, we subtracted the ambient power consumption of the system in its idle state from the measured values. For our configuration, the ambient power was around 145 W, and the saturation power value was close to 225 W, corresponding to an active range of roughly **80 W**. All measurements in this paper are with respect to this active range.

2.2 Experimental Results

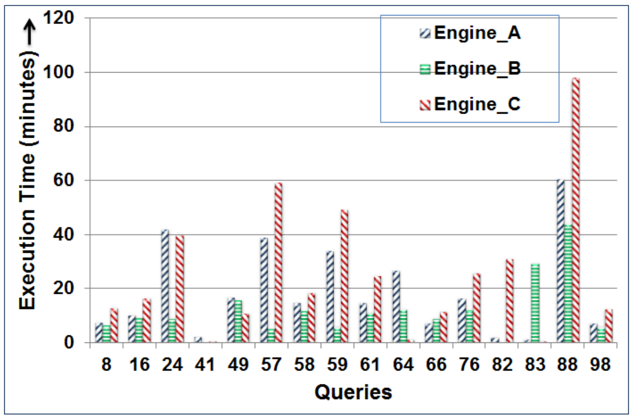
Under the ambit of the above experimental framework, we evaluated the peak power values obtained on the three database engines (EngineA, EngineB, EngineC) over each of the sixteen TPC-DS queries featuring in our workload. These results are presented in Figure 2.1(a). To provide the complete picture, we also provide the average power values and the query execution times in Figures 2.1(b) and 2.1(c), respectively.



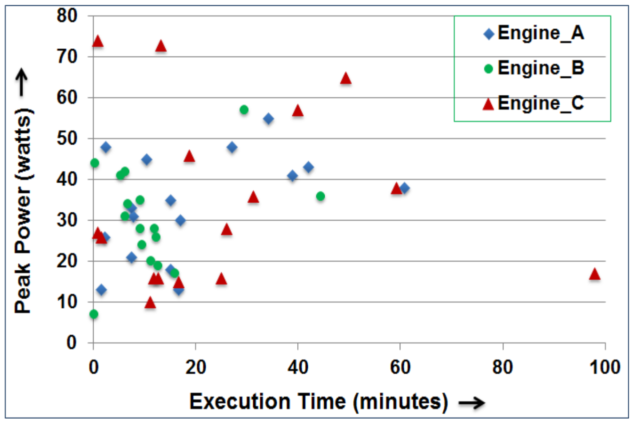
(a) **Peak Power**



(b) **Average Power**



(c) **Execution Time**



(d) **Power versus Time**

Figure 2.1: Power and Time Performance on TPC-DS Queries

From the Figure 2.1(a) we can see that, there exist queries covering the various categories that exercise the underlying computational platform through a substantial range of the 80W dynamic power limit. For example, with EngineA, about half-a-dozen queries (e.g. Q16) use more than 40W, 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 60W. Further, there are some queries, with Q41 and Q59 being prime examples, wherein *all* three engines incur high power requirements.

Interestingly, with both Q8 and Q64, the average power consumption is roughly similar across the three engines (Figure 2.1(b)) but their peak power behavior is very different (Figure 2.1(a)). This clearly demonstrates that peak power behavior cannot be easily correlated with average power characteristics, and several such instances are present in Figures 2.1(a)-2.1(c).

The above results gives sufficient evident to show that there is a material need to study and address the peak power consumption of database engines. Our first task was to assess the distribution of the power consumption over the various hardware resources – as mentioned earlier, we did this through an approximate temporal correlation of the operating system's resource utilization charts with the power meter readings. Based on this analysis, we found that virtually all the peak power consumption occurs at the *processors* and the *main memory*, whereas the disk overheads are negligible in comparison. These results are also corroborated in the recent work of [24], and we therefore only incorporate CPU and memory-related parameters in the prediction model proposed in the following chapter.

Chapter 3

Modeling Peak Power

A peak power estimator algorithm for query execution plans is proposed in this chapter. Our approach for the same is optimizer-agnostic but we restrict our attention to modeling EngineC for ease of presentation in the remainder of this thesis. Proposed algorithm is based on a regression model developed from a small set of heuristically chosen training examples. Inputs for the algorithm are solely based on information available in the plan descriptions provided by current optimizers. We hasten to add here an important caveat: 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 all the results presented in this thesis 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 execution of the sample queries). While this assumption is obviously untenable in practice, our objective here is to assess the innate quality of our estimation model.

Average power and peak power. Average power can be easily estimated by aggregating the estimates of energy consumption of each individual operator in the 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 sequence of operators, commonly referred to as "pipelines". In order to identify the pipelines present in a plan tree, we leverage the previous work on development of SQL progress indicators [5, 14] – in particular, we use the algorithm presented in [5]. As an example, the optimizer plan for benchmark query Q59 is shown in Figure 3.1(a), segmented into its constituent pipelines – here, there are 8 pipelines, PL1 through PL8 and there is an partial order of execution of pipelines enforced by blocking operators of the plan. e.g. PL4 can't start executing until PL3 is finished.

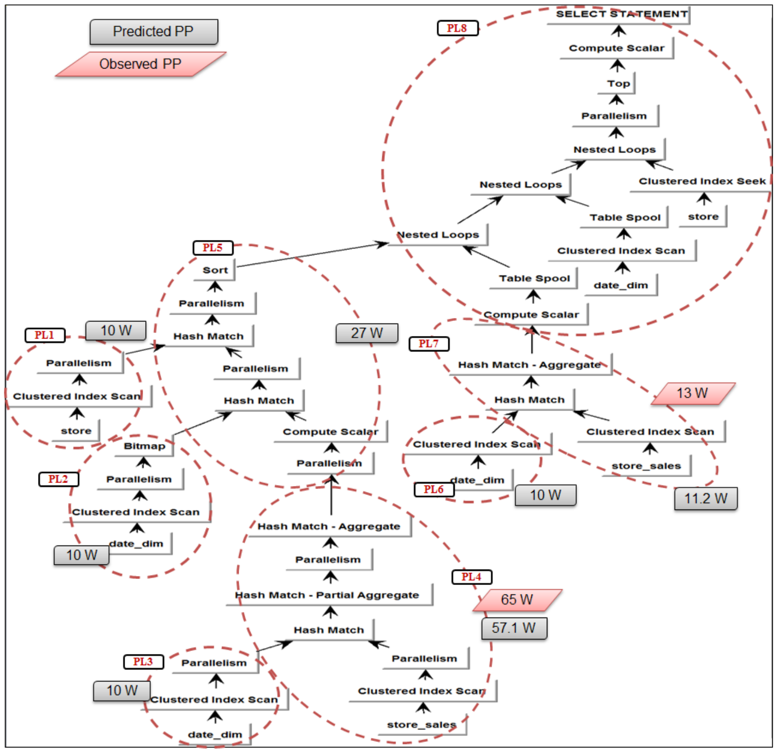
In Figure 3.1(a), the power figures in black on the various pipelines are the estimates from our model, and the peak power prediction for the entire execution plan is simply the *maximum* of these estimates – in this case, it would be 57.1 W. On the other hand, the power figures in red are the actual consumption at run-time, as determined from the temporal log of the power behavior during execution, shown in Figure 3.1(b). This attribution process is discussed later in Chapter 3.2.

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

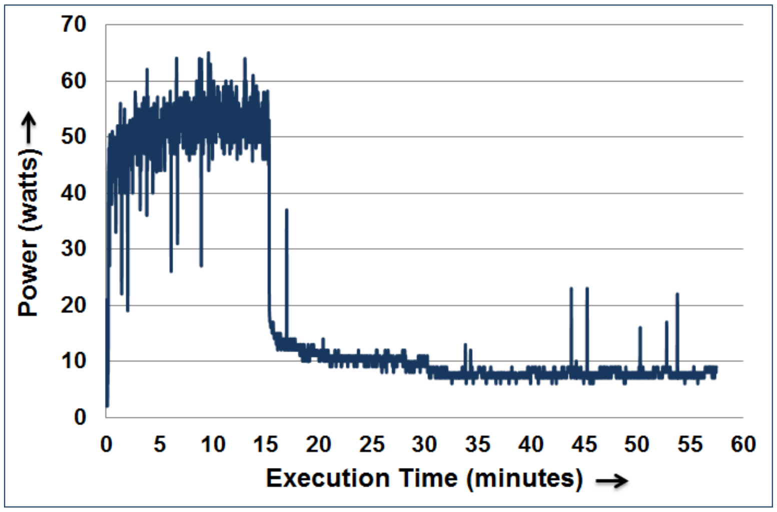
3.1 Pipeline Modeling

Each pipeline contains a set of *driver* nodes, comprised of the operators providing inputs to the pipeline, and a *termination* 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, is shown in Figure 3.2, which features in EngineC's plans for the Q8 and Q59 queries. This pipeline is driven by a `Clustered Index Scan` operator and is terminated by a `Hash Aggregate` operator. Since the build input of the `Hash Match` operator is itself blocking, this input is associated with a different pipeline that needs to be completed before execution of the ePL pipeline can commence.

From the 99 TPC-DS benchmark queries, we created a large number of variants with no two queries having same query plan, and collected statistics on the kinds of pipelines present in the associated optimizer plans. These results are shown in Table 3.1, where we see that although a large number (5765) of pipelines are present, most of them are structurally identical to others. Only 419 structurally-distinct pipelines were found in these plans. Further we removed power-significant operators, such as `Compute Scalar` from these distinct pipelines.



(a) **Pipeline Annotated Optimizer Plan**



(b) **Temporal Power Behavior**

Figure 3.1: TPC-DS Query 59

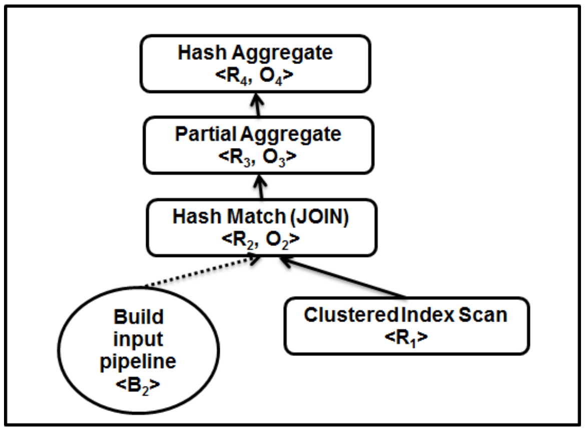


Figure 3.2: Example Pipeline (ePL)

The resulting pipelines are termed as *power-distinct pipelines* and are only 247 in number, and even among these, the number that appear frequently (more than 100 times) are only 10 but not all of these frequent pipelines are neccesarily power hungery pipeline.

Total Number of query plans	585
Total Number of pipelines	5765
Number of structurally-distinct pipelines	419
Number of power-distinct pipelines	247
Frequent power-distinct pipelines	10

Table 3.1: Pipeline Analysis of Query Execution Plans

3.1.1 Model Parameters

We now discuss our choice of regression model parameters. For ease of presentation, we make a distinction 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, which usually corresponds to a base relation, and (b) *Internal Pipelines*, wherein all inputs are from intermediate relations, that may be hosted on disk or are purely memory-resident. Our running pipeline example, ePL in Figure 3.2, is an example of a leaf pipeline.

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 estimated time for transferring this data from the disk. That is, for a driver node D, the rate is computed as

$$Rate_D = \frac{Input_D}{DiskTime_D} \tag{3.1}$$

where $Input_D$ denotes the size of data retrieved from disk by D and $DiskTime$ is the disk transfer time (for EngineC, these values are computed from *EstimateRows* * *AvgRowSize* and *EstimateIO* plan annotations, respectively).

With the above formulation and given a pipeline PL, the rates of the downstream operators in PL are derived using the formula shown in Equation 3.2. 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.

$$\begin{aligned}
\text{Let } Driver_N &= Subtree_N^{\mathsf{PL}} \cap Driver^{\mathsf{PL}} \\
\text{Then } Rate_N &= \frac{\sum_{i \in Source_N^{\mathsf{PL}}} Output_i}{\max_{x \in Driver_N} DiskTime_x}
\end{aligned} \tag{3.2}$$

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 ratio 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* 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.

Internal Pipelines. For internal pipelines, the driver nodes are the blocking termination nodes from *other* pipelines. There are two possibilities here:

- 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 3.2.
- The outputs produced by *all* the driver nodes are small enough to be fully memory resident, in which case the internal pipeline reads 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 only size-based model parameters for such pipelines – specifically, the input size to each operator in the pipeline. And as an effect of this, entire modeling of these pipelines need to be carried out separately.

Parameter Example. Consider again the ePL pipeline shown in Figure 3.2. 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 described in Table 3.2, and shown in the operator annotations of Figure 3.2. Specifically, each operator has an associated input data rate; in addition, the Hash Match has an input data size, while the two aggregates have output sizes, amounting to 7 parameters in toto.

Table 3.2: Candidate Parameters for ePL

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

3.1.2 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, we estimate that 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 [7]. 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 [16] 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 [15], we *incrementally* carry out the sampling using LHS, stopping stop 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 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. For example, to model the ePL pipeline, we used the following strategies to generate the training instances:

- The size of the scanned relation was altered to vary R_1 .
- 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.
- The join conditions were altered to vary R_3 without affecting R_2 and R_1.
- Various aggregates were added or modified to vary O_3, O_4 and R_4 without affecting R_3.

3.1.3 Regression Model

For all our pipelines, we use *stepwise* multi-linear regression models, which are recommended when there are several candidate explanatory variables, and no pre-defined theory on which to base the model selection [25, 26]. A further motivation to opt for the stepwise technique was that our candidate parameters are not independent features – for example, the various rates in a pipeline are correlated. A beneficial side-effect is that over-fitting of the model on the training data is also reduced in this approach.

We used the XLSTAT statistical analysis software [34] to fit our training data in a stepwise regression model. As a case in point, for ePL, we executed the 26 training queries and from the associated execution plans, created the training data with all seven parameters along with the observed peak power value for every query. The final model retains only *four* parameters, as shown below:

$$
\begin{aligned}
PeakPower(\text{ePL}) = 1.25 \times 10^{-6} \times R1 \\
+7.75 \times 10^{-6} \times R3 \\
+3.67 \times 10^{-6} \times R4 \\
-6.00 \times 10^{-10} \times X3
\end{aligned}
\tag{3.3}
$$

A graph of the observed values against the fitted values from Equation 5.1 is shown in Figure 3.3, with the dashed line signifying the ideal situation. It can be easily seen that all the training examples fall fairly close to the ideal, with the overall co-efficient of variation of the RMS error being only 0.13. The relative error between observed and predicted values is never more than 10% here.

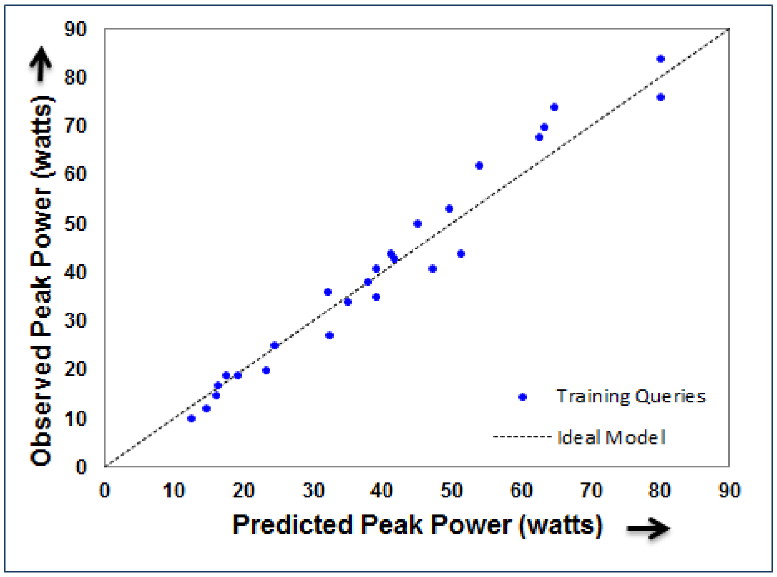


Figure 3.3: Regression Model on ePL

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

Table 3.3: Modeling Quality on ePL

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 10 W when the input rates are low, and upper bounded by 80 W when the inputs are very large and the CPU is fully saturated. Therefore, we add these bounds to our peak power estimator in Equation 5.1.

Modeling Accuracy. As mentioned earlier, ePL features in the optimizer plans for TPC-DS queries Q8 and Q59. We provide in Table 3.3 the model's prediction quality on ePL in these plans, where we see that the predicted values are in the neighborhood of the observed values.

3.1.4 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 3.4, 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:

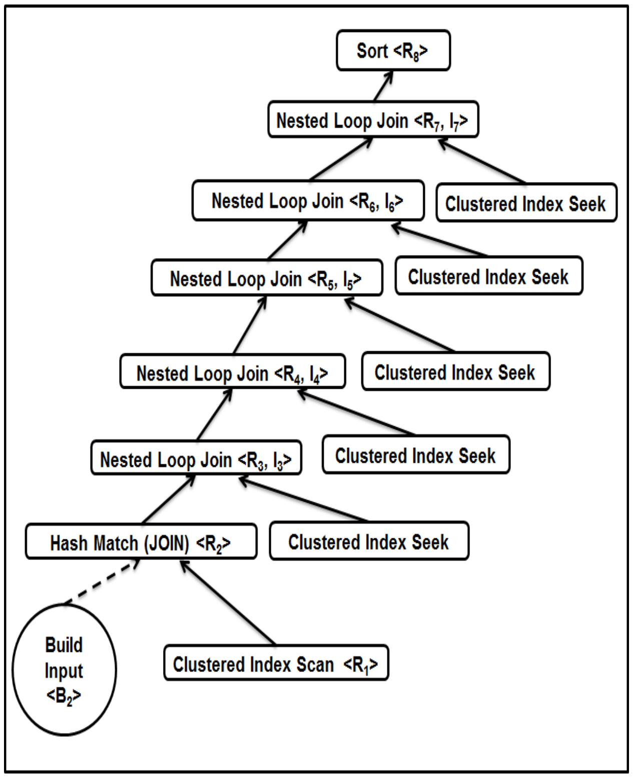

Sort <R8>
Nested Loop Join <R7, I7>
Nested Loop Join <R6, I6>
Clustered Index Seek
Nested Loop Join <R5, I5>
Clustered Index Seek
Nested Loop Join <R4, I4>
Clustered Index Seek
Nested Loop Join <R3, I3>
Clustered Index Seek
Hash Match (JOIN) <R2>
Clustered Index Seek
Build Input <B2>
Clustered Index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

Figure 3.4: Complex Pipeline cPL

$$PeakPower(\mathsf{cPL}) = 11.1 + 1.05 \times 10^{-5}\ R_3 + 1.02 \times 10^{-6}\ I_5 \tag{3.4}$$

Table 3.4 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 large pipelines consume only a modest amount of peak power.

Table 3.4: Modeling Quality on cPL

Query Number	**Peak Power**	
	Predicted (W)	**Observed (W)**
Q17	11.1	10
Q25	11.1	11

3.2 Results for Complete Plans

Thus far, we were discussing individual pipelines. We now move on to evaluating how our predictions perform on entire query plans.

Query Number	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 3.5: Predictions on Power-Intensive TPC-DS Queries

Reverting our attention to Figure 3.1(a), we show there the predicted peak power values 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 one-second granularity. Further, due to our black-box environment, in order to measure the actual peak power for a pipeline, we had to manually look through the temporal power log and approximately identify the time segment of its execution. Due to the complex 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 boxes of Figure 3.1(a). As can be seen, the predicted values (57 W and 11 W) are in the ballpark of the observed values (65 W and 13 W).

To generalize the above example, we show in Table 3.5 the summary set of prediction results for all the TPC-DS queries in Figure 2.1(a) on which EngineC consumed significant power – specifically, in excess of 30 W. In this table, we see that the predictions are consistently *within 15 percent of the observed values*, indicating that the model is, to the first degree of approximation, 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.

Chapter 4

Power Efficiency through Alternative Plan Choices

The results of the previous sections highlighted that database queries often trigger high-power bursts during the course of their execution. 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 "dynamic voltage scaling" (A detailed discussion is given in Section 6). 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 power-efficiency 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.

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. An example query template, QT59, derived from Q59, is shown in Figure 4.1), 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).

On this query template, we produce a "plan diagram" [21], 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) [12]. As a case in point, the 2D plan diagram corresponding to the QT59 template is shown in Figure 4.2, 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.

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. We show a graph of peak power against response time for Q59 with some plans from the POSP set in Figure 4.3. Observe that there is one plan: P8(green color) whose peak power, 45W, is significantly lower than that of the optimizer's plan choice (red color), 65W. Interestingly, in this case, the candidate replacement plans also happen to be much *more time-efficient* than the optimizer's choice – 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.

The replacement plan P8 is shown in Figure 4.4(a). It is segmented into pipelines and annotated with predicted peak power values. Also the temporal power log is given in Figure 4.4(b). It can be seen that our model accurately predicts the peak power and thus will be able to suggest replacing the original plan (given in Figure 3.1(a)) with P8.

We show in Figure 4.5, which captures peak power against response time for Q65, yet another situation where power-and-time efficient replacements can be identified. Here, there are replacement plans available which reduce peak power consumption substantially (by about 35 W) incurring a time penalty of less than 10%.

4.1 Power Diagrams

We further leveraged the query template plan diagrams by not only evaluating all the alternative plans at the original query location (corresponding to (100%,100%) selectivities in the diagram), but also at *all* the other locations in the parameter space.

Here we present results for QT59 (given in Figure 4.1) on the SQL Server. There are 47 plans in the plan diagram generated by Picasso at a resolution of 100*100. The plan diagram over 25 locations is shown in Figure 4.6(a). The diagram contains 10 different plans as can be seen. We then executed all these plans at all 25 locations and noted the peak power consumed during the execution. The plans were then replaced with peak power optimal plans at each location and a new plan diagram was formed which is given in Figure 4.6(b). The number of plans is reduced to only 4 in this diagram; these 4 plans are clearly peak power efficient in majority of the space and thus replace other plans.

```
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 4.1: Query Template 59

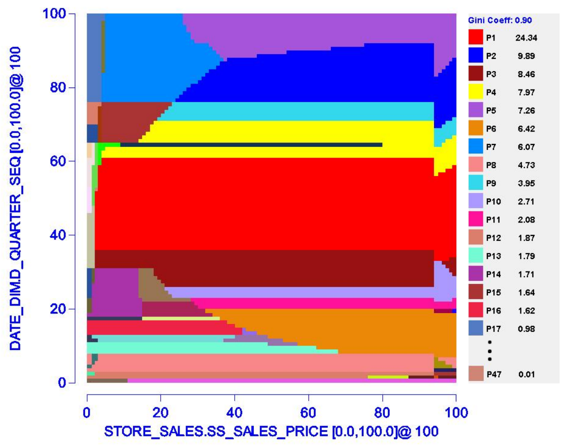


Figure 4.2: Plan diagram of QT59

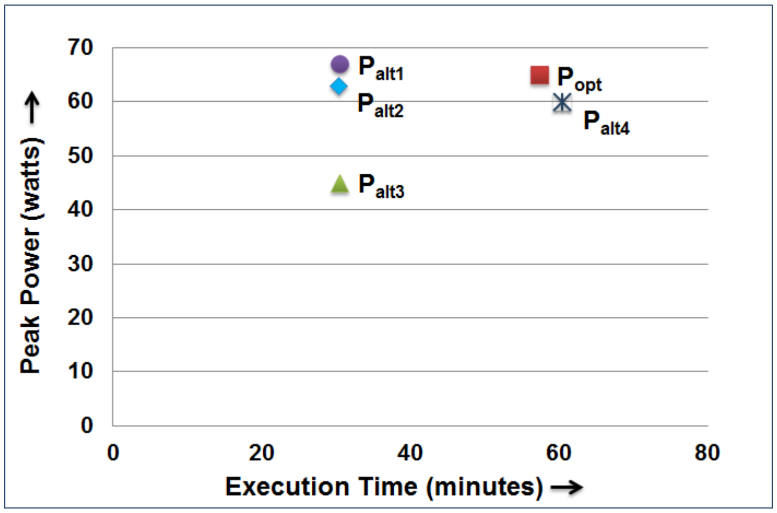
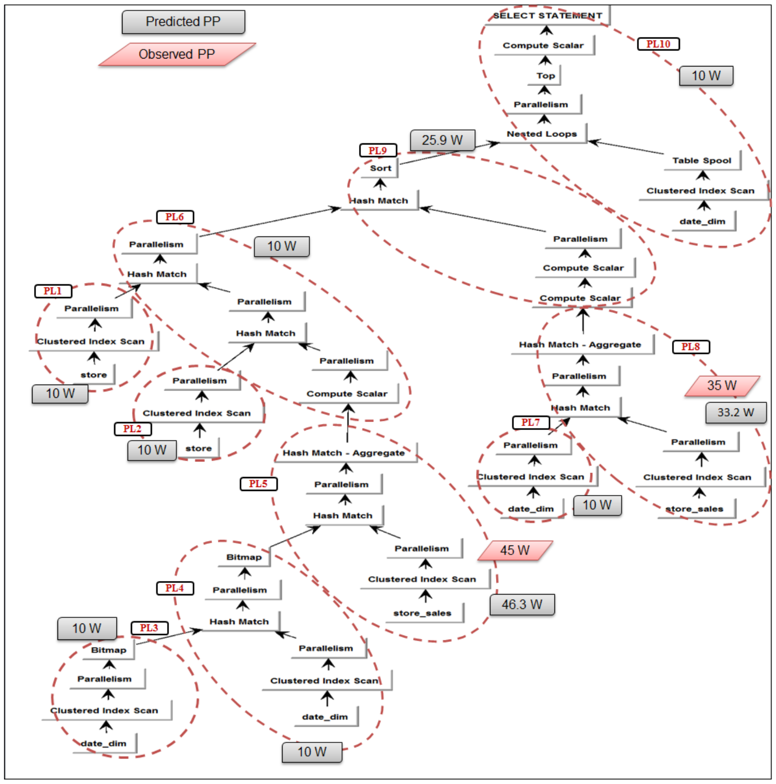
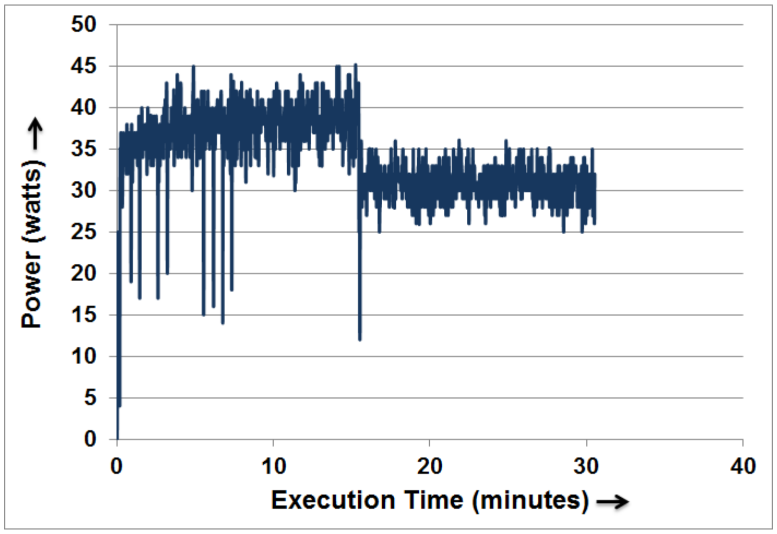


Figure 4.3: Peak power against Response time for Q59



(a) **Pipeline Annotated Optimizer Plan**



(b) **Temporal Power Behavior**

Figure 4.4: Plan P8 of Query 59

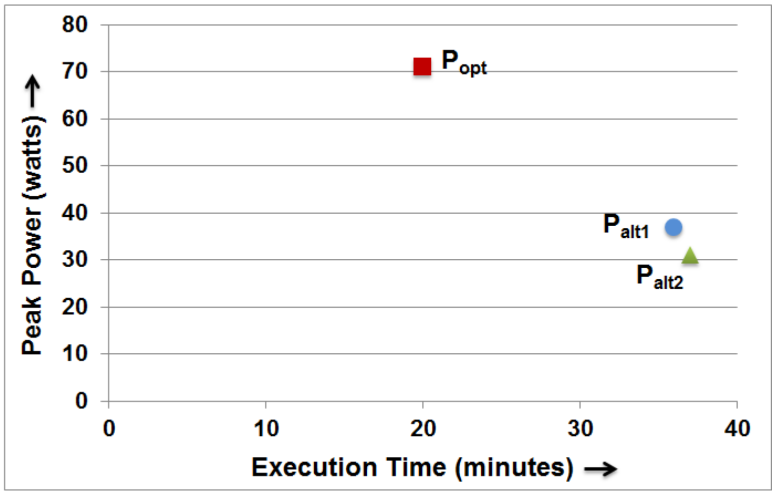
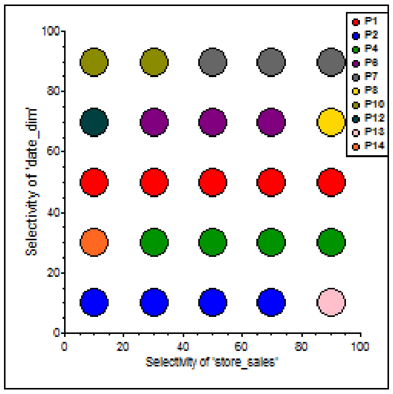
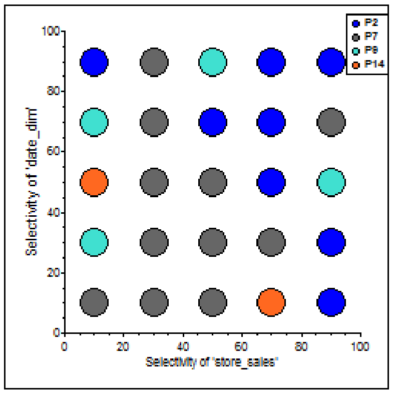


Figure 4.5: Peak power against Response time for Q65



(a) Compile-time plan diagram



(b) Peak power optimal plan diagram

Figure 4.6: Plan diagrams of QT59

Chapter 5

Modeling Extensions

In the previous chapters, 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.

5.1 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. Our preliminary assessment of this issue has yielded promising results in the case of hash-join sequences, as explained next.

We initially analyzed a pipeline with a sequence of two hash joins terminated by a sort operation, coming up with the following model, where O_1 and O_2 are the output sizes of the two hash joins:

$$PP_2 = 45.384 + 5.556 \times 10^{-5} \times R_2 - 1.302 \times 10^{-4} \times R_1 \tag{5.1}$$

This model was then generalized to the case of $n+1$ $(n \geq 2)$ hash-joins through the *recurrence* shown in Equation 5.2:

$$PeakPower_{n+1} = K \times PeakPower_n + C_{n+1} + A_{n+1} \times R_{n+1} \tag{5.2}$$

where O_{n+1} denotes the output size of the additional hash join, C_{n+1} and A_{n+1} being the associated parameter coefficients; $PeakPower_n$ 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 much fewer than 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.

Based on the above approach, the following equations were developed for pipelines with sequences of 3, 4 and 5 hash-joins (with PP used as shorthand to denote PeakPower):

$$PP_3 = -26.214 + 0.71 \times PP_2 + 8.326 \times 10^{-4} \times R_3 \tag{5.3}$$

$$PP_4 = 2.28 + 0.71 \times PP_3 + 9.28 \times 10^{-4} \times R_4 \tag{5.4}$$

$$PP_5 = 26.714 + 0.71 \times PP_4 + 2.01 \times 10^{-4} \times R_5 \tag{5.5}$$

The prediction quality of each of these recurrence-based models is shown in Table 5.1, for a variety of test-cases. We observe from the results that the relative error is always within 15%. Further, for reference purposes, the accuracy of the associated native model is also given in Table 5.1. We observe that the accuracies of both models are comparable, while the training overheads for the inductive model are considerably lower than those of the native model.

5.2 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, and an interesting research problem is to investigate how the single-query

Table 5.1: Inductive Modeling Accuracy

Test Query	**Inductive Prediction (W)**	**Observed Peak Power (W)**	**Native Prediction (W)**
3 Hash Join Pipeline			
Test1	36	42	34.8
Test2	53	47	42.6
Test3	10	12	11.8
4 Hash Join Pipeline			
Test1	49.7	50	46.1
Test2	10	10	13
5 Hash Join Pipeline			
Test1	60.8	61	53.7
Test2	16.3	18	17.3
Test3	40	47	43.2

models could be extended to accurately capture multi-query environments.

To assess the above, we carried out exploratory experiments with two concurrent queries. Our initial results suggest that if the queries are “data-disjoint”, that is, they do not share inputs, then 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 is because only a single pipeline is in execution at any given time in EngineC, independent of the number of concurrent queries.

On the other hand, if the two queries *share* part of their inputs, then some of the 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 start at staggered time instants. Therefore, we assume the worst-case – that is, where all input-sharing pipelines are modeled as memory-resident internal pipelines instead of leaf pipelines. With this assumption, we found that we were always able to provide an upper bound on the actual peak power consumed during the concurrent execution.

5.3 Incorporation in Engine

As mentioned previously, we have treated the database engines as black boxes, using only the functions available in their query processing APIs for developing our models. The limitations of this outside-the-engine approach are that we are never precisely sure about when one pipeline ends execution and another begins, or as to which operators are executing concurrently – in a nutshell, all our attributions are through inference. It would therefore be instructive to implement the model within a public-domain database engine such as PostgreSQL and assess how the additional information that becomes available to such an implementation can be leveraged to further improve the models.

5.4 Incorporation in Optimizer

We could also envisage a scenario where the query optimizer is given a specific *peak power budget* and asked to produce a time-efficient query execution plan that adheres to this budget. There are two approaches that could be taken here:

- Produce the most time-efficient plan as usual, and then identify whether one or more pipelines within this plan would violate the specified power budget. If so, then replace only the violating pipelines with acceptable greener alternatives.

- Use the power budget as a pruning mechanism for the enumeration carried out during the standard dynamic programming exercise to find the best plan, and thereby ensure that the final plan output by the optimizer automatically adheres to the budget.

Chapter 6

Related Work

Over the last decade, issues related to power and energy consumption have attracted considerable attention from the research community, fuelled by technological advances, environmental concerns and mobility considerations. The prior literature can be classified into two broad categories: (a) Hardware techniques intended for developing power-conscious computational platforms, and (b) Software techniques intended for programming power-conscious systems/application software. In this chapter, we review the salient highlights of this related work, covering power-aware database systems in most detail.

6.0.1 Hardware Techniques

In the early work of [13], hardware idleness was monitored by the operating system and a policy of powering down currently unused components was shown to provide significant power savings. Similar OS-driven energy reduction techniques, such as fine-grain adjustment of the clock speed, were discussed in [27, 29]. Making application server farms energy efficient through techniques such as dynamic voltage scaling have been analyzed through simulation in [8]. Compiler-directed schemes for reducing cache power consumption by turning off idle memory modules, and by modifying data transfer patterns across the memory hierarchy, are presented in [17]. Recently, [19] carried out a detailed survey to identify the most power-intensive components of data centers, and recommended strategies to reduce this burden through changes in the hardware components and their organization.

6.0.2 Software Techniques

Many applications possess a variety of execution paths to choose from in order to accomplish a given computational task, and software techniques that determine how applications can adjust their behavior according to the power-state of the underlying systems have been developed during the last few years. The need for incorporating power-awareness in database systems was keenly recognized and highlighted in the 2008 Claremont research vision report [2]. A variety of recent projects, primarily arising in industrial research labs, address such power-related issues. For example, an external sort benchmark called Joule-Sort was developed in [22] for holistically evaluating the energy efficiency of computer systems, with the metric being the number of records sorted per joule of consumed energy. The energy-performance tradeoffs of a variety of spatial access methods in memory-resident databases, typical of mobile systems, is profiled in [4]. Extensive empirical results on power consumption patterns in commercial database platforms are reported in [19]. A simple peak power model is built by summing *name-plate* power values of all the components, and this model is used to analyse power performance trends on TPC-C benchmark environment [32]. A common theme in all these prior efforts is that their attention is primarily focused on leveraging hardware opportunities, whereas our work attempts to analyze the problem from the software perspective.

In the last couple of years, the re-engineering of database optimizers to gain power efficiency has been increasingly viewed as a promising approach. For example, software developers are challenged in [10] to develop energy-efficient databases through reworking optimization choices, scheduling algorithms, physical database design and database update techniques. These thoughts were echoed in the visionary views of [11] wherein experimental evidence was 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 some explicit work on modifying the database query optimizer to choose more energy efficient query plans. Interestingly, the first such attempt was in [3], almost two decades ago. Here, the goal is 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 is assumed, their emphasis is 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 [18]. 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 indicated a good potential for improvements but these results are yet to be validated on real systems.

The study of average power behavior in database query optimizers presented last year in [28], 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. 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 [24]. Interestingly, and contrary to [28], they found that the best performing configuration is the most energy efficient configuration as well. Our own experience has also been the same, corroborating these results. Finally, they also highlight the potential for using software mechanisms to cap peak power consumption of database systems, and our work attempts to substantiate these views in a quantitative manner on industrial-strength platforms.

Chapter 7

Conclusions

We have investigated here, for the first time, the peak power behavior of modern database engines when processing complex SQL queries. Our "black box" study of a representative set of popular 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 quite 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 some carefully chosen sets of training examples. Our initial experimental results indicate that this model, which only uses generic plan parameters as inputs, is reasonably accurate in its predictions. It can therefore be a useful tool in designing database engines that robustly handle worst-case scenarios.

Through the development of "power diagrams", we also demonstrated how parametric optimal plan sets often throw up power-efficient alternatives to the optimizer's original choice without materially compromising the query running times. This observation serves to encourage the design of query optimizers that organically include power characteristics as a selection metric during their exploration of the plan space.

References

[1] Lamport, L. LaTeX: A Documentation System, Addison-Wesley Publishing Company, 1986.

[2] R. Agrawal et al, “The Claremont report on database research”, *Proc. of SIGMOD* 2008.

[3] R. Alonso and S. Ganguly, “Energy Efficient Query Optimization.”, Technical report, Matsushita Info Tech Lab, 1992.

[4] N. An et al, “Energy-performance trade-offs for spatial access methods on memory-resident data”, *The VLDB Journal Vol. 11*, 2002.

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

[6] C. Chekuri, W. Hasan, R. Motwani, “Scheduling problems in parallel query optimization”, *Proc. of PODS* 1995.

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

[8] E.N. Elnozahy, M. Kistler and R. Rajamony, “Energy-efficient server clusters”, *Proc. of PACS* 2002.

[9] 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.

[10] G. Graefe, “Database servers tailored to improve energy efficiency”, *Proc. of SETMDM* 2008.

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

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

[13] Y.-H. Lu and L. Benini and G. De Micheli, "Operating-system directed power reduction", *Proc. of ISLPED* 2000.

[14] G. Luo, J.F. Ellmann and M.W. Watzke, "Toward a progress indicator for database queries", *Proc. of SIGMOD* 2004.

[15] A. Matala, "Sample Size Requierement for Monte Carlosimulations using Latin Hypercube Sampling", Mat-2.4108 Independent Research Projects in Applied Mathematics, Helsinki University of Technology, 2008.

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

[17] O. Ozturk and M. Kandemir, "Energy management in software-controlled multi-level memory hierarchies", *Proc. of GLSVLSI* 2005.

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

[19] M. Poess and R.O. Nambiar, "Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results", *Proc. of VLDB* 2008.

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

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

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

[23] D.S. Soper, "The Free Statistics Calculators Website", 2011. Online Software, `http://www.danielsoper.com/statcalc/`

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

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

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

[27] M. Weiser, B. Welch, A. Demers and S. Shenker, "Scheduling for reduced CPU energy", *Proc. of OSDI* 1994.

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

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

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

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

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

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

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