EXPLORING OPPORTUNITIES OF POWER SAVINGS IN DATABASE QUERY OPTIMIZERS

A Project Report
Submitted in partial fulfilment of the
requirements for the Degree of
Master of Engineering
in
Computer Science and Engineering

by

Mayuresh Kunjir

Computer Science and Automation
Indian Institute of Science
BANGALORE – 560 012

JUNE 2010

TO

My Parents

Acknowledgements

I would like to thank my advisor Prof. Jayant Haritsa for allowing me to pursue this project under his guidance. I thank him, for his constant and valuable guidance, support and encouragement during my stay at IISc.

I would also like to thank my project partner Mahesh Bale and other members of Database Systems Lab for providing a stimulating and fun environment for work. I would like to thank my other friends and Gymkhana club members who have made my stay at IISc memorable.

I thank my parents for their continued support throuout my career.

Abstract

As the energy consumption in computing systems is increasing, much attention is being paid to make computing systems more power efficient. While there has been a lot of work done in the past in designing better hardware or hardware control mechanisms for this purpose, there has not been much work in making software applications power-aware. Since database management systems are a major component of computing systems, energy consumption by them is also significant and hence there is a need to make them power-aware. There hasn't been much study done to check the current state of the database query optimizers in terms of power efficiency. Since the goal of present query optimizers is to minimize the response time, it is very much possible that energy or peak power consumption is being compromized.

In our work, we try to see if this is the case with current database optimizers. We explore all the existing opportunities to improve on these power metrics and evaluate the possible improvements. The analysis to understand the trade-off between energy/peak power and performance has also been carried out.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the years, the main objective of computing system designers has been improving performance. But with the increasing energy consumption of computers, power efficiency has become a critical issue in system design. A recent report by the Environmental Protection Agency shows that data center power consumption in the US doubled between 2000 and 2006 [2]. In a typical data center, electricity consumed by servers and cooling systems contributes to around 20% of the total ownership cost, equivalent to one-third of the maintenance cost. This makes energy the second largest item in a typical IT department's monthly bill (with labour being number one). It is generally believed that these numbers will continue to increase in the next few years. Uncontrolled energy usage in the data centers has negative implications on the environment. Due to the continuous demand for increase in the computational capabilities and the continuous decrease in size for portability, the modern data servers suffer from an increasing possibility of thermal failures. Therefore, it is important to reduce the database power consumption so that the total power consumption of an entire system can be kept below a given power budget. Furthermore, reduced power consumption can lead to lower system temperature implying reduced cooling costs and this can contribute to significant energy savings in the long run.

Due to the above mentioned facts, power conservation in computing systems has attracted much attention from government agencies, industrial sectors, and the research communities. Standards and benchmarks are being modified to consider power consumption [16, 18]. Transaction Processing Council(TPC) is also developing a new energy specification [17]. Early innovations on power-aware computer servers have concentrated on hardware and system software like compilers and operating systems. These build a computing environment where: (1)Hardware can operate on various power saving modes [6]; (2)System software can control the operating modes of hardware and reschedule resource requests from applications to save energy [9, 11]. These system level research approaches treat the high level applications as passive resource consumers that are not aware of the low level efforts of power reduction. As a complementary measure, we need to build power aware applications which can exploit the power saving opportunities in system.

While software for mobile(battery-powered) devices may include techniques targeted at energy efficiency, current server software typically does not. For database servers, some techniques aimed at general performance and scalability like *data compression* may also have incidental benefits in power efficiency, but designing database servers specifically for energy efficiency is not yet attempted. Power reduction in DBMSs is of high economic significance as DBMS is an important type of software in the three-tier computing architecture adopted by most of today's business computing environments. In a typical data center, a majority of the computing resources are dedicated to database servers, making DBMSs the largest consumers of power in all the software applications deployed. It is the need of hour to study and analyze the power consumption by commercial DBMSs and try to find out if they are overlooking power in the quest to achieve quick response time.

In our work, we study power consumption patterns in database management systems and identify power saving opportunities. The main objective is to find if there are any power-saving opportunities in current DBMSs and if there are, see how we can exploit them. We have identified two metrics : *Energy* and *Peak Power* which should be minimized for the DBMS. We included peak power metric as to our knowledge, there hasn't been any published work that analyzes peak power consumption of DBMS and we believe this metric presents good potential for improvement. Our ultimate goal is to build power aware DBMS by changing query optimizer that can achieve significant energy/peak power cost savings while maintaining reasonable performance in query processing. We focus on the opportunities provided by the database query optimizers. Power savings can also be achieved by re-designing the database layout or the access algorithms but our aim is to make query optimizers power-aware and this will serve as a compliment to the other efforts. We use a popular commercial database engine and PostgreSQL[1] database engine to analyze power consumption patterns of different query plans considered by optimizers.

Contributions

In our quest to find out power saving opportunities provided by the optimizers, we found out some interesting results. The contributions of this work are as follows.

- Through our analysis of database optimizers we found out that there do not exist alternate strategies that reduce energy consumption significantly and the few cases where we do get improvement,

[1]`http://www.postgresql.org/`

it is limited and not more than 10% of the optimizer's choice.

- The peak power interestingly, provides scope for improvements in commercial database engines. We were able to find out alternate strategies giving about 50% improvements here. Although it is difficult to model peak power usage, this is an area that warrants attention from the optimizer developer community.

Organization

As mentioned before, we execute different query plans given by optimizer and analyze power consumption by them. Typically, database engines execute only the optimal plan chosen by optimizer. We first discuss in Section 2 how we achieved *foreign plan execution* in the two engines we are working on. Next we give details of our setup and workload in Section 3. Following this, we move towards finding opportunities of power saving in database optimizers. Section 4 is dedicated to energy metric while Section 5 talks about peak power. An overview of related work is given in Section 6 and finally Section 7 summarizes all the results and also concludes the work.

Chapter 2

Foreign Plan Execution

The goal of our project was to find the opportunities of power saving in the database optimizers. As we know, an optimizer considers a lot of plans/strategies to execute the given query and selects the one which it thinks will give the best performance (i.e. execute the fastest). Since our goals of optimization are different (energy and peak power), the best performing plan may not be the best for us. But to verify this, we need to execute the other plan(s) than the optimizer's choice and this feature is not directly provided by any of the database engines. Commercial engines do provide some ways to achieve this with facilities like query hints and abstract plan execution which will be discussed in Section 2.1. PostgreSQL engine though doesn't provide any such facilities and hence we modified the plan selection procedure of Postgres optimizer to choose a subset of plans from the search space those can be executed by the engine. First we will see how we achieved foreign plan execution in SQL Server and then see how to explore search space in PostgreSQL.

2.1 Foreign Plan Execution in SQL Server

Microsoft SQL Server engine has a utility called *Use Plan Hint* using which we can provide a plan in XML format along with a query and if that plan happens to be from the search space (i.e. a valid plan considered by the optimizer), the engine executes the query with this plan even if it is not the optimal one. Now the only question that remains is how do we know that the plan we are providing is a valid plan from the search space. To ensure that it is, we move our attention to a subset of plans called *Parameteric Optimal Set of Plans* (POSP). Next paragraph gives some definitions before showing how to use this subset.

For a query on a given database and system configuration, the optimizer's plan choice is primarily a function of the selectivities of the base relations participating in the query : that is, the estimated number of rows of each relation relevant to producing the final result. A plan diagram [24] is a color-coded pictorial enumeration of the execution plan choices of a database query optimizer for a parameterized query template over the relational selectivity space. The dimension of a plan diagram is the number of relations whose selectivities are varied to produce the picture. Figure 2.1 below shows a sample plan diagram for a Query Template built from Q22 of TPC-DS benchmark suite. It is a 2-dimensional plan diagram drawn at a resolution of 100*100. The minimal set of plans that includes at least one optimal plan for each point of the selectivity space is called Parameteric Optimal Set of Plans (POSP) [22]. The POSP for the figure has 14 plans.

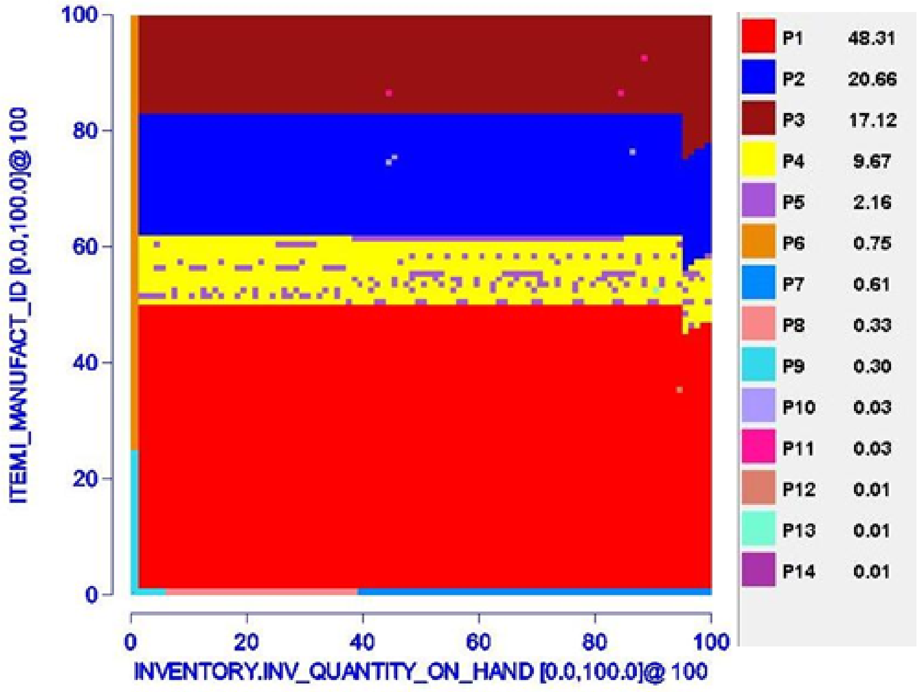


Figure 2.1: Plan diagram of QT22

We used the Picasso[23] tool to generate the plan diagrams. All the plan diagrams generated for our experiments were two dimensional. The plan diagram contains all the plans from the POSP at a certain degree of resolution. For each of these plans, we generated abstract XML plans from the SQL Server. These abstract plans were executable at any point of the selectivity space since they were guaranteed to be from the search space of the optimizer and thus we could achieve the foreign plan execution (although for a limited number of plans) in the SQL Server.

2.2 Plan Selection in PostgreSQL

The planner unit of Postgres engine follows bottom-up approach for plan selection [28]. It considers all possible execution paths for the given query and through dynamic programming selects the one which is predicted to execute the fastest. The plan tree is constructed for this optimal plan and is passed to executor unit which executes it. We wanted planner unit to give away more plans and not to pass only the optimal one so that we can see the energy consumption pattern for all of them. One way is to modify the whole process of plan selection and instead of rejecting some paths at each level, consider all of them. But there are two problems here: (1)There would be too many plans reaching the top level, (2)Most of these will give very bad response time and won't do any better in terms of energy as well. Our goal was to find out a subset of plans from the search space which are not much worse than the optimal plan.

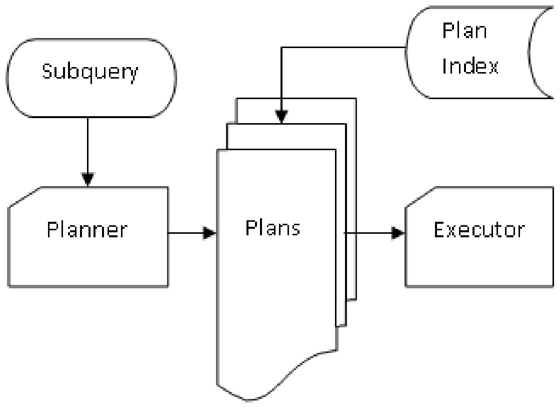


Figure 2.2: Modified plan selection in PostgreSQL

A query can contain a number of sub-queries because of nesting of queries or union/join of multiple parts. The optimization for each sub-query is carried out separately by Postgres. For each sub-query, once a plan reaches top level of optimization, only the grouping and/or aggregation operators remain to be considered by planner which are to be carried out in the end. The plans that reach up to this level are better in terms of performance than the other discarded ones and one of these is chosen in the end which performs the best after applying grouping/aggregate. We modified the planner unit to pass all these plans[1] instead of just the optimal one. This set of plans is called as *Root Node Space* of the query. The flow of our plan selection process is shown in Figure 2.2. We added one extra parameter that indicates the index for each sub-query. Out of all the plans passed by the planner unit, the executor unit selects one using this index. So by varying this index, we can execute different plans. These plans

[1]The number of plans here were observed to be between 6-10 for the TPC-DS benchmark queries.

are not necessarily the best performing ones because it is possible that some of the better plans are pruned by the ones that reach top. This problem is discussed in [26]. But ranking all the plans of the search space is a tedious job and for our experiments, the plans reaching the top level seem to form a good representative set. In the future, if the plan ranking feature is added to Postgres engine, we would like to use it for our tests.

Chapter 3

Experimental Setup

3.1 Server Configuration

We used a Sun Ultra-24 workstation having a quad-core processor as our database server. The configuration is given in the Table 3.1.

Workstation	Sun Ultra 24
Processor	Intel Core 2 Extreme Quad Core: 3.0 GHz
RAM	8 GB
Hard Disks	4*300GB SAS: 15000 RPM
Operating System	Windows Vista Business: 64 bit

Table 3.1: Database Server Configuration

Two database engines were used for all our experiments. A commercial engine : Microsoft SQL Server 2008 and an open source engine : PostgreSQL 8.4. Both are widely used DBMSs.

3.2 Workload

In all our experiments we used the TPC-DS[1] benchmark which is the new generation industry standard decision support benchmark. It models the decision support functions of a multi-channel retails product supplier. The schema is highly complex with a total of 25 tables. There are three sales channels : *store*, *catalog* and *web* each of which has one *sales* and one *returns* table. These six along with *inventory* data constitute the fact tables of the schema. The other tables are mainly the dimension tables which store

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

critical business data like information about customers, orders etc. The scale 1 size of the database is 100GB and we used this sized database for our experiments. The benchmark also has a set of 99 highly complex queries. Its workload is expected to test the upper limits of hardware system performance in the areas of CPU, memory and I/O subsystem utilization and the ability of the operating system and database software to perform various complex functions important to DSS as given below.

- Examine large volumes of data
- Give answers to real-world business questions
- Execute queries with a high degree of complexity and various operational requirements (e.g. ad-hoc, reporting, data extraction)

3.3 Power Measurement

We used a digital power meter model 20-1850/CI manufactured by Brand Electronics [15] for our power measurements. The meter has to be connected to an electricity supply outlet and it has a socket to which our database server was connected. This way, the meter could record the power drawn by the server. The meter has a computer interface cable which has a serial port connector at the other end. Through this cable it can transmit the measured power values. We logged the power values of our server using this in a *monitor* machine at a sampling rate of 1Hz. For each logged power value, the time stamp at which it was measured was also logged. The scripts to calculate energy, average power and peak power values from the recorded logs were written on the monitor machine and were used in all our experiments.

Chapter 4

Plan Replacements for Energy Conservation

In this section, we try to find if there exist any plans which are superior than the optimizer's choice in terms of energy. As mentioned before, we do this on a commercial engine SQL Server as well as on an open source engine PostgreSQL. The results for each are given in separate subsections below.

4.1 Plans from POSP on SQL Server

Here we chose some query templates and created the compile time plan diagrams for them using the Picasso tool. Next we considered all the plans from POSP which can be executed at any location of the selectivity space using the 'Use plan hint' feature as mentioned in Section 2.1. We then executed these plans at 25 equally spaced positions on the selectivity space and measured power values using our power measurement setup. At each of these points, we replaced the optimal plan with the one that is most energy efficient and tried to find how much improvements are possible. We also tried to see trade-off between the energy value and the performance to know how much penalty in terms of performance we have to pay in trying to save energy.

4.1.1 Detailed Analysis for a Query Template

As mentioned in the experimental setup, we carried out the plan replacements for energy optimality on some query templates over TPC-DS data. We show the results for Query Template 22 (given in Figure 4.1) here. The plan diagram generated using Picasso for this template with diagram resolution 100*100 contains 14 different plans. (shown in Figure 2.1.) As mentioned previously, we only consider 25 equally-spaced locations of the selectivity space. The compile-time optimal plan diagram for these

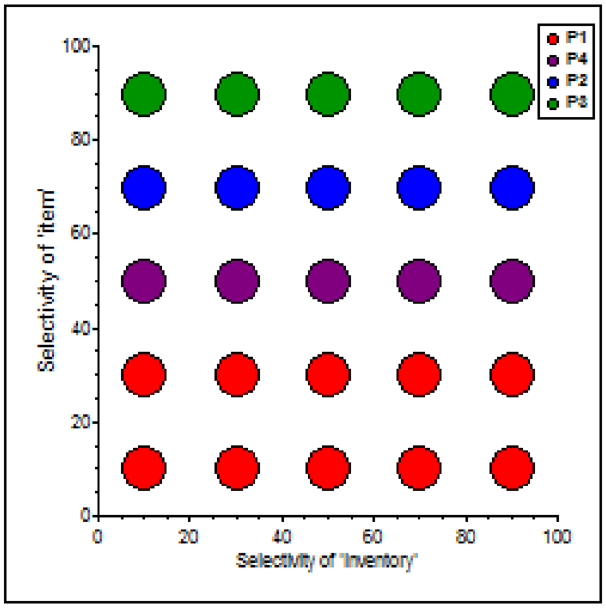
```
select top 100 i_product_name
             ,i_brand
             ,i_class
             ,i_category
             ,avg(inv_quantity_on_hand) qoh
       from inventory
           ,date_dim
           ,item
           ,warehouse
       where inv_date_sk=d_date_sk
         and inv_item_sk=i_item_sk
         and inv_warehouse_sk = w_warehouse_sk
         and d_year=1998
         and inv_quantity_on_hand :varies
         and i_manufact_id :varies
       group by rollup(i_product_name
                      ,i_brand
                      ,i_class
                      ,i_category)
       order by qoh, i_product_name,
                i_brand, i_class, i_category;
```

Figure 4.1: Query Template 22

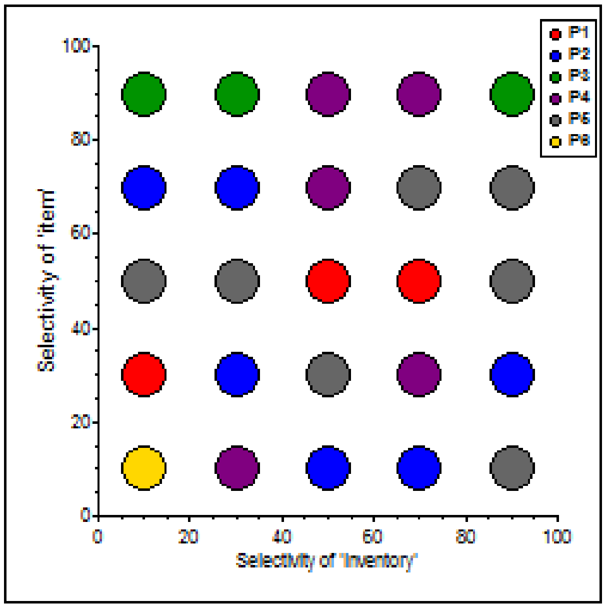
25 points is shown in Figure 4.2a. At each of these points, we executed all the 14 plans from POSP and noted down the energy values. We then replaced the plan at each location with a plan that is the most energy efficient at that location. The new diagram thus achieved is given in Figure 4.2b. It can be observed that at only 6 locations out of the 25, original plans are retained. Though plans change at most of the locations, we need to find the magnitude of the change.

Figure 4.3 shows the energy cost diagram for both compile-time optimal plan diagram and energy (POSP-)optimal plan diagram where the third dimension shows the value of the energy consumed for all the points of the selectivity space[1]. It can be seen that at most locations, there is very small difference between the two curves. The energy optimal plans save only 8.5% energy over the optimizer's plan choices over the whole selectivity space. We can see two spikes in the curve of compile-time optimal energy cost diagram. We tried to see why this happened and found out that this is in fact due to suboptimal choice of plan by optimizer in these locations. The optimization errors due to wrong cost estimation are common and an accepted fact in database literature [21]. If optimizer had selected correct time-optimal plan in these two locations, the energy savings would have come down to only 3% which is a very small value.

[1]The diagram is interpolated across the 25 points we have.



(a) Compile-time plan diagram



(b) Energy optimal plan diagram

Figure 4.2: Plan diagrams for QT22

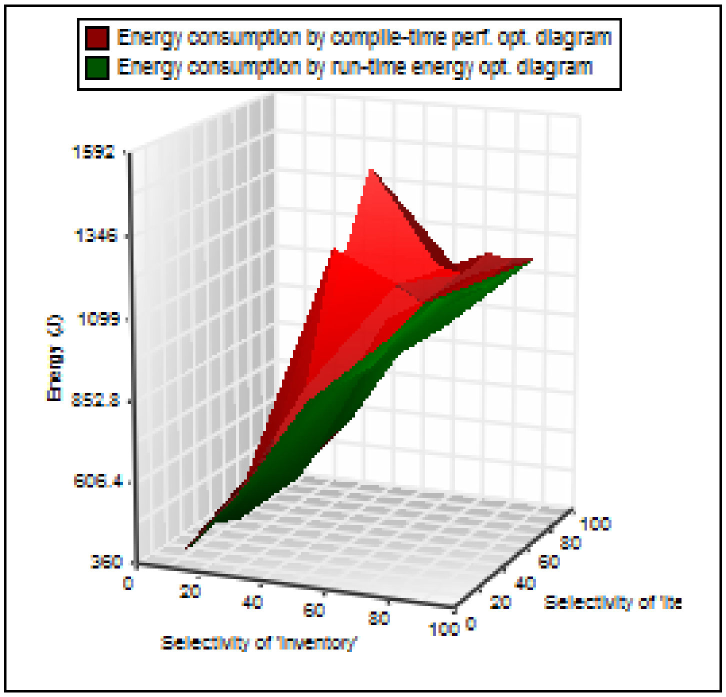


Figure 4.3: Energy cost diagram for QT22

We found similar results for QT8 as well which are not included here. (It showed only 1.7% improvement.) So it doesn't seem to be worth the effort to change the whole database optimizer to gain these miniscule savings in energy.

4.1.2 Trade-off between Energy and Performance

We showed in the last subsection that there are replacement plans available from POSP that are estimated to save on energy (though the savings are not substantial). In order to see how worse they are in terms of performance, we executed all the plans from POSP for some query templates on the same database engine. Figure 4.4 shows the results for the templates executed at the top-right point of the selectivity space. Here different points for a query stand for different plans from POSP. The times and energy values for all the queries are normalized to the range [0, 1] to fit them all in a single graph. It can be observed that for every query, some plans appear near origin and some far off in top-right area. The later ones are worst plan choices both in terms of time and energy. There are some plans for Q65 which are more energy efficient than optimizer choices, but here we get about 8% energy savings for a performance penalty of more than 17%. For other queries, we do not find any plan doing better than the optimizer's choice. So all this signifies that the optimization done for performance alone suffices to provide energy optimal choices in most cases.

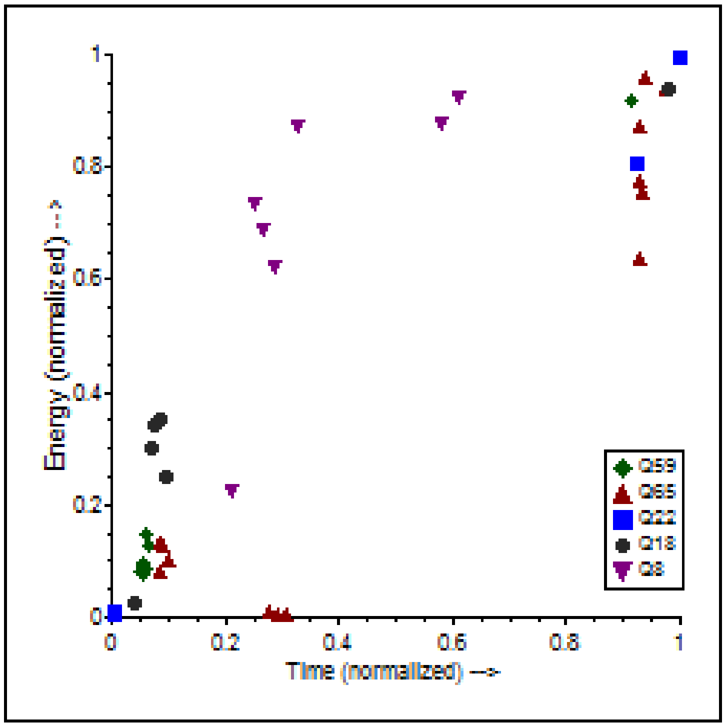


Figure 4.4: Energy against performance for some benchmark queries

4.2 Energy Model on PostgreSQL

In the last subsection, we explored the POSP space and found that there are not many opportunities existing in the query optimizer to save energy of a query. Here we try to dig into the search space of the optimizer by first building a simple energy model and then trying to see if we can identify any superior plans. All these experiments were carried out on the PostgreSQL 8.4 database engine.

The work here is inspired from [4]. It claims that by modifying the Postgres optimizer, we can achieve power savings in the range of 11% to 22%[2]. It develops a power cost model along the lines of Postgres time cost model by adding some constants that assign power costs for basic operations like accessing a tuple in CPU. For a given workload, it estimates the average power value for every plan and selects the one with the minimum value. There is a serious flaw with this model though. The model uses some formulae to estimate the power cost which are similar to the ones used by Postgres default optimizer to estimate the time cost and this logically doesn't seem correct. To elaborate further, consider the case of sequential scan. As given in [4], the power cost function for sequential scan suggests that the CPU power is directly proportional to the *number of tuples* of the relation that is being scanned. But the average power should remain the same irrespective of the number of tuples, only the time cost should change. We tried to execute some sequential scan queries by varying the number of tuples and found that the average power is roughly the same for all the queries, only the energy values changed due to the variation in the execution times.

As our focus is on energy, we decided to develop a model to estimate the energy values directly and not the average power. We came up with a very simple model to estimate the energy for a given query. We next discuss the model, the workload we chose for our tests and the results obtained.

4.2.1 Our Model

We decided to build the energy model on top of Postgres cost model. Postgres provides us with the time cost estimates and our job is to come up with good energy estimates. We know that CPU and disks are the units which consume dynamic power[3]. The power consumption by other units doesn't vary once it is stabilized after system boot and it is accounted for in the ambient/idle power value. The dynamic peak power consumed by CPU in our configuration is about 80W and that taken by the disks is 4W[4]. Postgres estimates the time cost taken by CPU and disk separately and the final estimate is

[2]These savings are shown on average power values for some specific workloads and no significant improvements are observed on the energy.

[3]Memory units also consume dynamic power, but that value is very small and leakage power tends to dominate the power consumption there.

[4]The values are measured using 'LEM Analyst 2060 Clamp-on Power Quality Meter'.

the sum of these two values. We define two constants for our energy model:

- CPU_TUPLE_POWER
- PAGE_POWER

Since our CPU is quad-core and a tuple is processed by a single core, we set the value of CPU_TUPLE_POWER to 20W and the other constant, PAGE_POWER which specifies disk cost is set to 4W. Finally the energy estimate is given by following formula:

$$\begin{aligned} Energy &= CPU\_TUPLE\_POWER * CPU\_time \\ &+ PAGE\_POWER * disk\_time \end{aligned}$$

where CPU_time and $disk\_time$ are the estimates given by the Postgres cost model.

Our model is dependent on the two parameters mentioned above and depending on the system configuration, these parameters need to be tuned. Basically it just scales CPU costs and disk costs differently since the power consumption by these units is different.

4.2.2 Workload

As mentioned before, we carried out all our experiments on TPC-DS database. The workload for the experiments is made up of TPC-DS benchmark queries. These 99 queries were classified into four categories according to the schema coverage in [27]. The classification is given below.

- Single fact table (54 queries)
- Multiple fact table

    with join of sub-queries (22 queries)

    with union of sub-queries (17 queries)
- Dimension table only (6 queries)

We created a workload of 16 queries chosen from these different categories with the proportion of the queries maintained[5]. These queries exhibit a wide range of SQL features ranging from huge aggregates to SQL CASE statements.

[5]Specifically, we selected 9 queries from first category, 3 each from second and third category and 1 query from last category

4.2.3 Experiments and Results

We executed each of the 16 queries with Postgres default optimizer as well as with our modified optimizer and compared the energy values as well as the performance. The results are presented in the Table 4.1. Though we were expecting to get energy efficient plans with our model, surprisingly only 7 queries showed any improvements the in energy values after execution. These improvements were not more than 6% anywhere which is very small. Remaining queries gave very bad results, 4 queries got such bad plans that they failed after executing for long amount of time. Some errors are bound to happen due to inherent weaknesses in modelling but the results here indicate the possibility that there may not exist any superior plans available in the whole search space. We tried to dig more into the search space of Postgres optimizer to find out the reasons for this behaviour. Next subsection talks about those experiments.

Query No.	Default Optimizer		Energy Optimizer	
	Exec. Time	Energy(J)	Exec. Time	Energy(J)
8	0:07:08	7029	0:07:25	7012
16	7:17:59	1137363	Failed	
24	0:12:00	16975	Failed	
41	0:06:13	13524	0:06:13	13524
49	0:32:21	6462	0:19:27	43382
57	0:05:54	9214	0:06:00	8877
58	0:13:35	13049	0:13:38	12775
59	0:32:50	78783	0:31:57	78246
61	0:13:52	13687	0:14:44	15459
64	0:24:43	34234	Failed	
66	0:06:49	8061	0:07:41	9523
76	0:13:53	14837	0:14:14	14290
82	0:02:24	4159	0:04:25	4554
83	0:01:09	1137	0:01:01	1097
88	0:55:06	57317	Failed	
98	0:06:41	6484	0:06:54	6077

Table 4.1: Comparison of Default optimizer and Energy optimizer

4.3 Exploring Root Node Space on PostgreSQL

As seen in our experiments with building energy model, the improvements in the energy are very limited. There are two possibilities here:

- The energy model is very weak and it cannot find the energy optimal plans from the search space.
- There are no energy optimal plans existing in the whole search space, so building the energy model is a fruitless exercise.

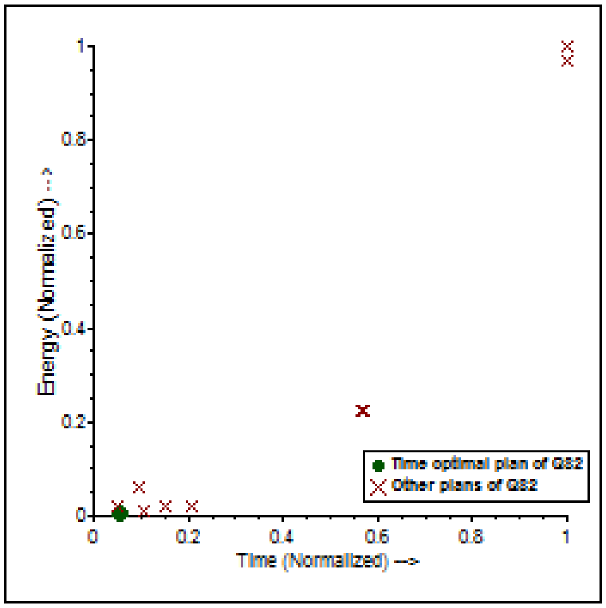
We changed the Postgres engine to allow it to execute a query plan other than the optimal one chosen by the optimizer. These changes are described in Section 2.2. Here we give the results we got with the changed optimizer and analyze those.

We tried executing the plans for some TPC-DS benchmark queries using the foreign plan execution feature discussed earlier. We discuss results of two queries here. One query is CPU intensive while the other one is I/O intensive. Figure 4.5a displays the trade-off between energy and performance for Query 82 of the benchmark set which is I/O intensive. This query is a join of 4 relations and involves only single fact table. We got 10 plans at the top level which we executed by changing the plan index as mentioned in Section 2.2. The energy and the execution times were recorded and the graph was plotted. It can be seen that the time optimal plan is the energy optimal one as well. There are 5 other plans which are closer in the energy value but no one beats the time optimal plan.

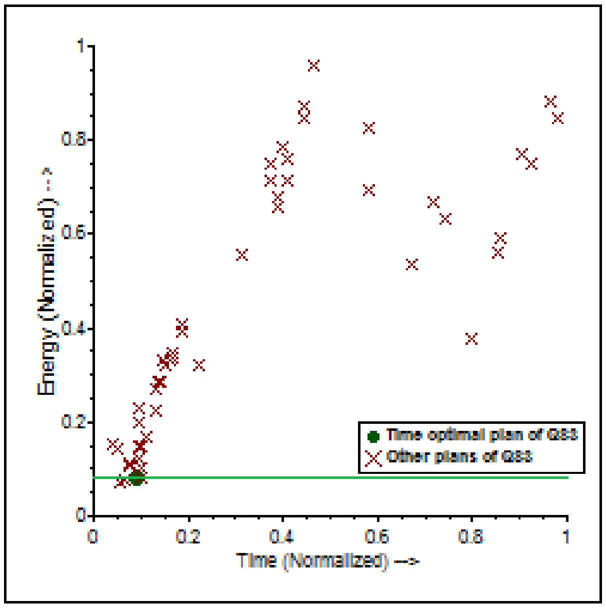
Figure 4.5b shows similar analysis for Query 83 which is CPU intensive. This is a complex SQL query and involves a join of all three fact tables. It consists of 4 sub-queries and each sub-query had 6-8 plans reaching the top level. So we had a wide range of plans available for the final query and the compilation costs of all these plans were within 25% of the cost of the optimal one. We executed all these plans and the graph shows the results for 52 of those plans[6]. Here we found only two plans that are better in terms of energy than the time optimal one. The difference though is only 4.7% and if you observe carefully, both these plans are time efficient as well. So not getting energy optimal plan in the first place from the default optimizer again seems to be the artifact of the compile-time errors in time estimation.

The results here tally with our observations with the energy model. Our model couldn't find more energy efficient plan for Q82 because there didn't exist one in the subset of plans we saw here and for Q83, we were able to find a superior plan which is one of the two that we saw doing better.

[6]The other plans turned out to be very bad in terms of both performance and energy after execution and hence not included.



(a) Q82



(b) Q83

Figure 4.5: Energy against Performance in PostgreSQL

Chapter 5

Plan Replacements for Peak Power Optimality

Since we found no scope for energy improvements for current optimizers, we moved our attention to our second metric : *peak power*. Some systems cannot afford to have high peak power consumption [12], so we tried to check opportunities of reducing peak power usage. As in the last section, here as well we first carry out detailed analysis on POSP space and then on the whole search space.

5.1 Plans from POSP for SQL Server

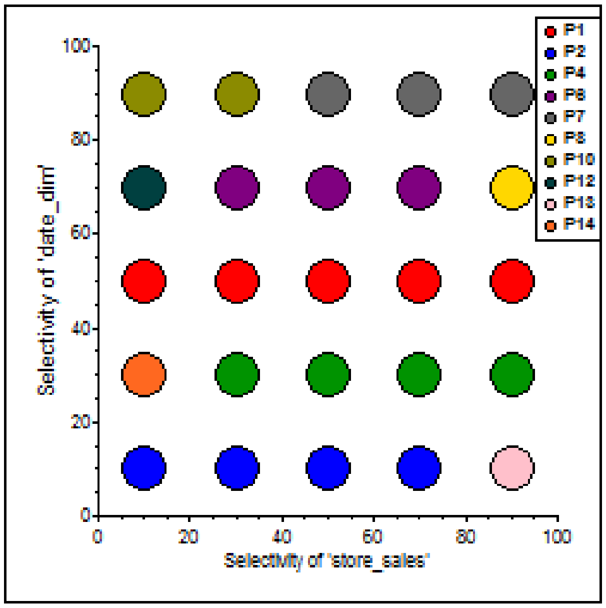
We first show detailed analysis for a query template as we did in the last section. Then we will do a trade-off analysis between peak power and time.

5.1.1 Detailed Analysis for a Query Template

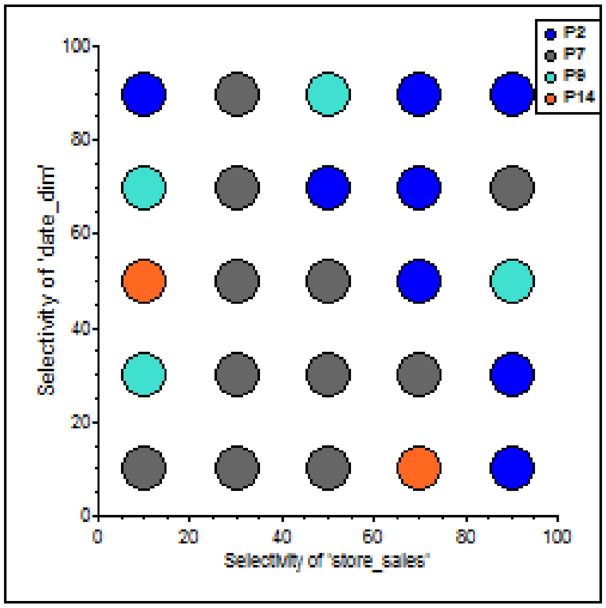
Here we present results for QT59 (given in Figure 5.1) on the SQL Server. There were 32 plans in the plan diagram generated by Picasso at a resolution of 10*10 and we again tried 25 equally spaced positions of the selectivity space. The plan diagram is shown in Figure 5.2a. The diagram contains 10 different plans as can be seen. We then executed all the plans from POSP on these locations and noted the peak power consumed during the execution. The plans were then replaced with peak power optimal plans and a new plan diagram was formed which is given in Figure 5.2b. The number of plans is reduced to only 4 in this diagram; these 4 plans are clearly peak power efficient in the whole space and thus replace other plans. But again the question is how much benefits are possible by doing this exercise.

```
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 5.1: Query Template 59



(a) Compile-time plan diagram



(b) Peak power optimal plan diagram

Figure 5.2: Plan diagrams of QT59

Figure 5.3 depicts the peak power values for the original compile-time plan diagram as well as the peak power (POSP-)optimal plan diagram. It is observed that at some points of the selectivity space, significant reduction in values is indeed possible. In fact at two points, we got reduction by as high as 43 Watts. This is really high considering the fact that the dynamic power range of our test machines is about 80 Watts. *To speak in terms of CPU utilization, we were able to bring down peak CPU utilization from about 90% to only about 25% by doing these plan replacements.* So peak power metric seems to hold potential for improvement.

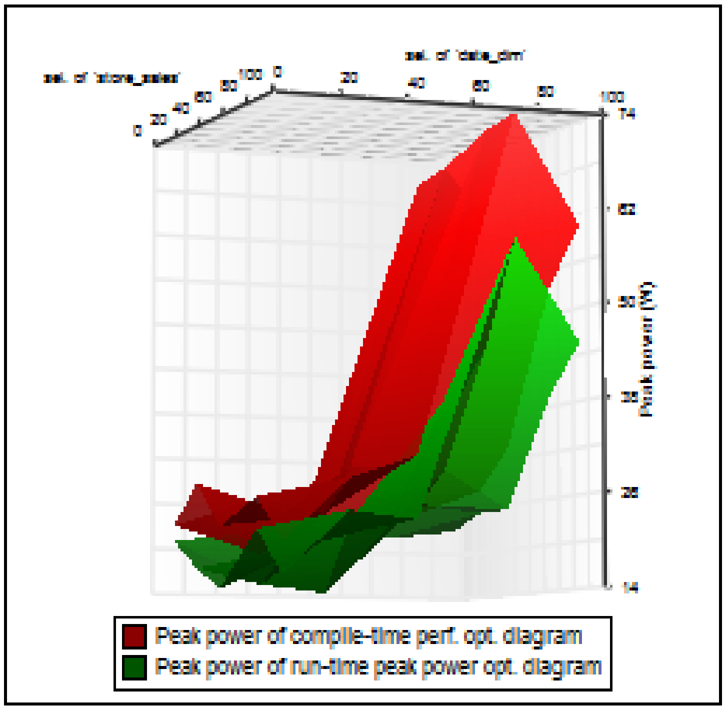


Figure 5.3: Peak power cost diagram for QT59

We observed an interesting aspect for this query template. The power behavior of plans over whole selectivity space is similar. We show the temporal power behavior of two plans P1 and P7 in Figure 5.4. The top row shows plan P1 for two selectivity points (50, 70) and (90, 90). For both the points, the graph shows high power usage[1] for about 970 seconds and then after a sudden spike, the power value drops down. The average power usage is seen higher for the higher selectivity point. Similar graphs for plan P7 are shown in the bottom row of the figure. This plan doesn't show the spike at the end of 970 seconds. This can be explained from the plan trees of these two plans. Both the plans compute a temporary view *wss* for the first 970 seconds and thereafter use this view to give the required result

[1]Note that the values of power in the graphs do not include ambient values and the power taken by database engine only is shown.

rows. The only difference is that the plan P1 uses a hash aggregate operation with parallelization at the end of the view computation which is a high power consuming operation and therefore the sudden spike in power appears. So from the peak power perspective, plan P1 is not good and therefore gets eliminated in the peak power efficient plan diagram.

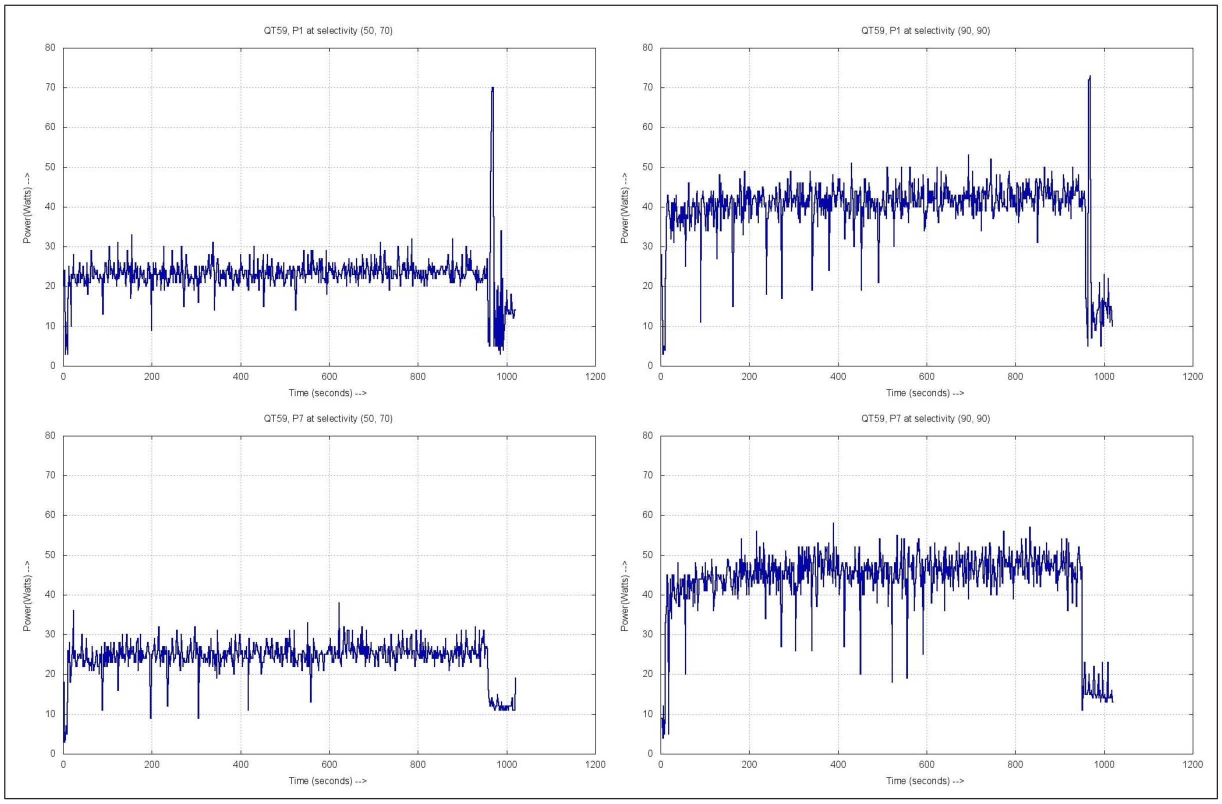


Figure 5.4: Temporal behavior of plans P1 and P7 at two selectivity locations

5.1.2 Trade-off between Peak Power and Performance

We just saw that there are plans available from POSP that provide significant reduction in peak power values. But how good are these replacements in terms of performance? When we looked at the time costs of these replacement plans, we observed that some of the replacement plans are worse by value as high as 150% of the original choice. The good news is that these cases were found at low selectivity points where we were getting very low absolute reductions in the range of 2-8 Watts. All the points at higher selectivity points were observed to have replacements available that don't increase the time cost much. We set a threshold of 30% on the increase of time cost i.e. we restricted the search space for replacements to only those plans of POSP that have time costs bounded by a value 30% more than the cost of the time optimal plan at that location. The plan diagram thus obtained is shown in Figure 5.5. The number of plans here is increased to 6, but the changes mostly occurred at low selectivity points and the points giving high reduction had their plans retained. The diagram showing peak power cost over selectivity space for this plan diagram is very similar to Figure 5.3 and is thus not included here. So this shows that *it is possible to achieve high benefits in peak power values with a small penalty in the performance.*

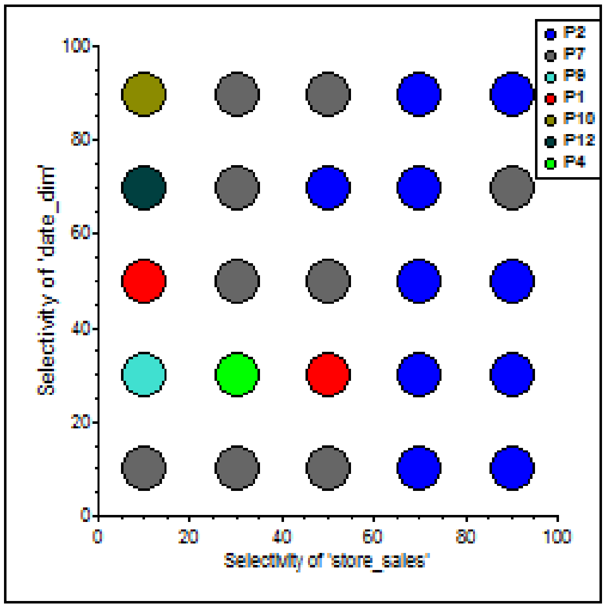


Figure 5.5: Peak power optimal plan diagram for QT59 with 30% threshold on performance

To verify our observations, we tried another query template QT65. Its observations at top-right corner point of selectivity space are given in Figure 5.6. The plans are seen to be clustered in three groups. One group is that of time optimal plans that seems to be very inefficient in terms of peak power usage. The second group at the right bottom is highly inefficient in terms of performance but seems to be doing well in terms of peak power. The interesting group is the third one which is very good in terms of peak power usage and also not doing too badly in terms of performance. In numbers, these plans bring down peak power usage from about 65W to 27W and that too with only 17% performance penalty.

So this commercial database engine clearly presents good opportunities for peak power savings from the POSP space itself and we don't even have to dig down into the whole search space.

5.2 Plans from Root Node Space on PostgreSQL

We also analyzed peak power behavior of the plans from Root Node Space on Postgres using the procedure described in Section 2.2. Unlike the commercial engine that we tried for our experiments in POSP space, Postgres doesn't show many fluctuations in peak power values. We didn't find much

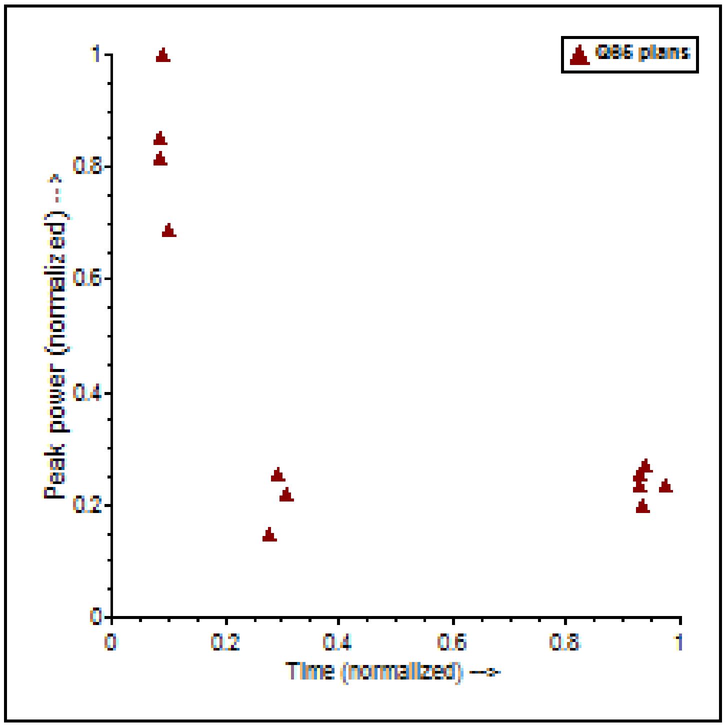


Figure 5.6: Peak power against performance for QT65

difference in the peak power values of different plans. Figure 5.7 shows the trade-off between peak power and performance for Query 83 of TPC-DS benchmark suite. The same plans that were tried for energy experiments were tried here. Only single plan is seen to be taking lower peak power than the optimizer's choice here and the difference is of 5W only. This behavior can be attributed to the lower complexity of the Postgres engine. It doesn't have complex operators like 'Hash aggregate with parallelism' used by SQL Server which consumes high power.

This analysis shows that the current commercial optimizers present the opportunities for peak power savings and our results indicate that the peak power value can be reduced substantially with a little performance penalty. The subset of plans that we tried on Postgres didn't show much improvements as mentioned above. The other plans of search space which we didn't try owing to their worse perfomance might give some different results. This is an area we would like to explore in the future.

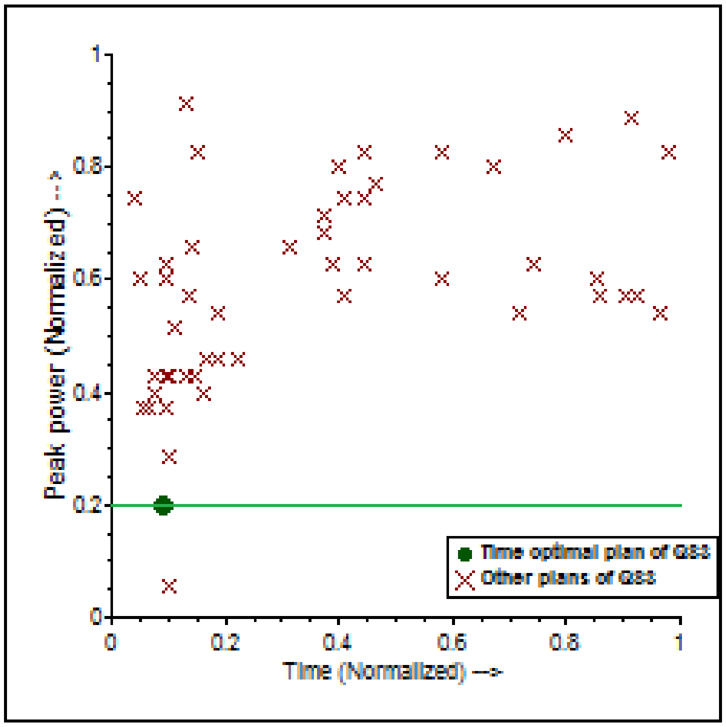


Figure 5.7: Peak power against Performance for Q83 on PostgreSQL

Chapter 6

Related Work

There has been a lot of research work to make systems power efficient by making some changes in hardware components; whereas the area of modifying softwares to make them power-aware is relatively new. We will cover some of the important work done both in hardware as well as software areas.

6.1 Hardware approaches

System-level power-related research has created an ocean of literature. [6] identifies hardware idleness and invokes a policy of shutting down unused components thereby getting up to 50% power savings. [7, 8] also talk about some changes in operating systems like adjusting clock speed at a fine grain to get savings in CPU energy. A lot of work has been done to manage the power-saving modes of hardware components. There has also been some work to make application servers energy efficient. [13] presents five policies for cluster-wide power management in server farms like dynamic voltage scaling. They use simulation to show savings of upto 29% on each server node. There has been some compiler directed schemes as well. e.g. [9] presents some ways to reduce power consumption in caches by turning memory modules on/off and modifying the data transfer patterns across the different levels of memory hierarchy. All this is done by compiler instructions and the scheme is simulated. Recently [3] did a survey to identify most power intensive components and suggest some approaches to make data centers more power efficient, all being some changes in the hardware or the organization.

6.2 Software approaches

Application-level power-aware computing has become an active research area as many applications have different execution paths to accomplish same computational task and we can make applications adjust their behavior according to power-related states of underlying systems.

The power consumption in databases has just caught attention of the research community. The Claremont report on database research [1] mentions "designing power-aware DBMSs that limit energy costs without sacrificing scalability" as an important research area. Some recent projects address power consumption issues in data centers. [18] develops an external sort benchmark for evaluating the energy efficiency of computer systems. To decide energy efficiency of system, the metric they use is the records sorted per joule of energy consumed and the machine that sorts maximum records is the most energy efficient. [3] reports extensive experimental results on the power consumption patterns in typical commercial database servers. It develops a power consumption model and verifies it with some standard TPC-C[1] configurations. In [14], energy behavior of mobile devices is analyzed when spatial access methods are used for retrieving memory-resident data. However, none of these above mentioned work suggests any ways of modifying database software and all focus on utilizing hardware opportunities only.

We focus on changing database query optimizer to choose more energy efficient query plans. The first attempt at such work was by [10]. It selects energy efficient query plans to increase effective battery life of mobile computers. Though this work looks very similar to our proposed area, their emphasis is on client-server model optimizing the network throughput and overall energy consumption. [11] suggests some energy management schemes for memory resident databases. Most of the ways suggested are hardware directed but they also give some query restructuring and regrouping algorithms which are implemented in PostgreSQL database engine. [20] presents a challenge to software and tool developers of developing energy-efficient databases by showing some good opportunities in hardware, optimization choices, scheduling algorithms, physical database design and update techniques. Recently [19] carries out some experiments to show that the current query optimizer's choice of plans is not necessarily the most energy efficient one. It suggests some energy-aware optimizations and configuration parameters and also emphasizes on the need to revisit and redesign database algorithms and policies. We found a recent work [4] as the closest to our proposed work. It first shows the opportunities in current database optimizers for power savings, then it modifies the query optimizer to take power costs into account. A power cost model is developed to find these costs which we have discussed in Section 4.2. More recently, [29] tried out different hardware and software configurations on some database engines and found that the best performing configuration is the most energy efficient configuration as well.

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

Chapter 7

Summary and Future Work

In this report, we tried to analyze the current state of the database engines in terms of power efficiency. We explored the search space of query plans for an open source database query optimizer and for a commercial database query optimizer, we explored a subset of query plans called as 'Parameterized Optimal Set of Plans'. Through our experiments on TPC-DS database, we found that nothing needs to be done for energy efficiency. The query plans chosen by the optimizers to maximize performance in fact maximize energy efficiency in most cases and the possible improvements are very limited. But this is not true in terms of peak power. In the past, some work has been done to look at the energy efficiency of the database systems, but to our knowledge nobody has tried to analyze peak power consumption of DBMSs. And as it turns out, the current commercial optimizers present opportunities to save peak power consumption by value as high as 50% and that too without affecting the performance significantly. This is an area that needs attention of the optimizer developer's community.

As we have shown that the database optimizers need to consider peak power metric while optimizing the queries, we would like to do it ourselves in the future by building a model to estimate the peak power usage and implementing it in a database query optimizer.

Appendix A

Query Templates

```
select top 100 s_store_name
      ,sum(ss_net_profit)
 from store_sales
     ,date_dim
     ,store,
     (select ca_zip
     from (
     (SELECT substr(ca_zip,1,5) ca_zip
      FROM customer_address
      WHERE substr(ca_zip,1,5) IN (
                          '99148','14140','53002','21895','12160','92371',
                          '14892','49798','15459','12274','70495',
                          '52014','97715','17739','10237','59188',
                          '83998','90166','25772','97130','18686',
                          '16144','66528','10939','89359','50546',
                          '30150','90878','92542','35479','26668',
                          '50744','33483','73506','61965','84608',
                          '82155','13303','30076','13262','90592',
                          '99213','11604','37243','58571','14265',
                          '36297','64689','23523','85050','50849',
                          '17720','41566','58749','44070','15193',
                          '68080','13213','14633','11509','28628',
                          '40857','79615','99340','43278','85381',
                          '97906','30365','24846','15285','64156',
                          '16183','15420','61890','89167','87448',
                          '86652','72778','96451','51110','26267',
                          '69191','63159','69961','27098','40363',
                          '55755','37145','23057','80486','45076',
                          '73748','37483','55719','90083','11111',
```

Figure A.1: QT8

```
'70719','71893','44138','68042','45137',
'23295','70904','66296','26446','20180',
'89283','21145','44909','75323','63215',
'21939','15023','42900','83907','47194',
'11242','84786','32397','64373','92503',
'23474','15737','50855','50307','65490',
'54288','14702','89635','84185','82133',
'72038','60421','67015','52932','89945',
'20501','60893','32215','71638','28918',
'79762','11218','57914','40119','36050',
'17258','73328','74624','67165','20258',
'71270','58847','77831','78860','34002',
'22724','92108','79474','70003','90826',
'87205','31724','44667','17204','88118',
'84555','39069','23272','64365','77008',
'40836','16061','52024','14535','82792',
'62101','70486','39463','28119','61632',
'28595','18978','56104','77779','85312',
'43500','94720','98734','99903','40381',
'28199','15003','49085','94399','69092',
'45352','45625','23283','39259','15304',
'45920','95140','97697','58585','46836',
'68927','88218','39686','11072','66324',
'39078','33940','55498','54573','37720',
'29019','84218','17756','99756','33661',
'45443','17385','29337','83251','53478',
'63093','76171','65602','87062','14513',
'58621','47249','86013','10326','68022',
'94174','60681','92429','64727','51405',
'18946','59123','48620','60710','87260',
'27761','81166','92700','51784','52410',
'18292','55938','46249','55973','34089',
'11436','84960','39659','28900','21903',
'41621','14730','35981','16052','38228',
'89731','12515','66129','59193','25836',
'43550','70205','13716','90747','20160',
'14761','71464','62258','49338','38468',
'41152','59074','27128','19560','84651',
```

Figure A.1: (continued)

```
                          '33965','70636','57842','20804','62718',
                          '83798','52370','66972','20575','72245',
                          '37496','84513','46109','27771','23169',
                          '18743','30983','62517','99350','78594',
                          '68692','91125','49710','12011','56381',
                          '15141','72848','57428','35931','70945',
                          '69971','66589','92723','31244','61082',
                          '71368','27655','16342','96700','64711',
                          '11505','25325','77784','84228','18003',
                          '33332','49845','17608','80083','42301',
                          '35907','45148','13535','21491','15566',
                          '20965','34297','17601','21942','12711',
                          '14371','11902','50493','64682','46051',
                          '51199','39617','94734','87962','66753',
                          '19329','35061','34874','15093','36999',
                          '20446','52042','87426','29565','57806',
                          '47719','27661','15507','16218','15595',
                          '16675','39953','74150','15571','84779',
                          '74361','79055','83043','36642','16477',
                          '27961','16752','71407','14629','21834',
                          '10113','74613','17133','32694','37181',
                          '67632','64728','55210','14911','84467',
                          '49413','92304','61489','56839'))
    intersect
    (select ca_zip
     from (SELECT substr(ca_zip,1,5) ca_zip,count(*) cnt
           FROM customer_address, customer
           WHERE ca_address_sk = c_current_addr_sk and
                 c_preferred_cust_flag='Y'
           group by ca_zip
           having count(*) > 10)A1))A2) V1
where ss_store_sk = s_store_sk
 and ss_sold_date_sk = d_date_sk
 and d_qoy = 1 and d_year = 2000
 and (substr(s_zip,1,2) = substr(V1.ca_zip,1,2))
 and ss_list_price :varies
 and d_quarter_seq :varies
group by s_store_name
order by s_store_name
;
```

Figure A.1: (continued)

```
select top 100 i_item_id,
        ca_country,
        ca_state,
        ca_county,
        avg( cast(cs_quantity as numeric(12,2))) agg1,
        avg( cast(cs_list_price as numeric(12,2))) agg2,
        avg( cast(cs_coupon_amt as numeric(12,2))) agg3,
        avg( cast(cs_sales_price as numeric(12,2))) agg4,
        avg( cast(cs_net_profit as numeric(12,2))) agg5,
        avg( cast(c_birth_year as numeric(12,2))) agg6,
        avg( cast(cd1.cd_dep_count as numeric(12,2))) agg7
 from catalog_sales, customer_demographics cd1,
      customer_demographics cd2, customer, customer_address, date_dim, item
 where cs_sold_date_sk = d_date_sk and
       cs_item_sk = i_item_sk and
       cs_bill_cdemo_sk = cd1.cd_demo_sk and
       cs_bill_customer_sk = c_customer_sk and
       cd1.cd_gender = 'F' and
       cd1.cd_education_status = '2 yr Degree' and
       c_current_cdemo_sk = cd2.cd_demo_sk and
       c_current_addr_sk = ca_address_sk and
       c_birth_month in (10,9,7,5,1,3) and
       d_year = 2001 and
       ca_state in ('CA','CA','WV'
                   ,'OH','UT','ID','VA')
       and cs_list_price :varies
       and i_current_price :varies
 group by rollup (i_item_id, ca_country, ca_state, ca_county)
 order by ca_country,
        ca_state,
        ca_county,
i_item_id
 ;
```

Figure A.2: QT18

```
select top 100 s_store_name, i_item_desc, sc.revenue
 from store, item,
     (select ss_store_sk, avg(revenue) as ave
  from
      (select  ss_store_sk, ss_item_sk,
       sum(ss_sales_price) as revenue
  from store_sales, date_dim
  where ss_sold_date_sk = d_date_sk and d_year = 2000
             and ss_list_price :varies
  group by ss_store_sk, ss_item_sk) sa
  group by ss_store_sk) sb,
     (select  ss_store_sk, ss_item_sk, sum(ss_sales_price) as revenue
  from store_sales, date_dim
  where ss_sold_date_sk = d_date_sk and d_year = 2000
  group by ss_store_sk, ss_item_sk) sc
 where sb.ss_store_sk = sc.ss_store_sk and
       sc.revenue <= 0.1 * sb.ave and
       s_store_sk = sc.ss_store_sk and
       i_item_sk = sc.ss_item_sk and
       i_current_price :varies
 order by s_store_name, i_item_desc
;
```

Figure A.3: QT65

Appendix B

Other Queries

```
select top 100 i_item_id
       ,i_item_desc
       ,i_current_price
 from item, inventory, date_dim, store_sales
 where i_current_price between 65 and 65+30
 and inv_item_sk = i_item_sk
 and d_date_sk=inv_date_sk
 and d_date between cast('2000-02-05' as date) and
(cast('2000-02-05' as date) +  60 days)
 and i_manufact_id in (526,89,748,463)
 and inv_quantity_on_hand between 100 and 500
 and ss_item_sk = i_item_sk
 group by i_item_id,i_item_desc,i_current_price
 order by i_item_id
 ;
```

Figure B.1: Q82

```
with sr_items as
 (select i_item_id item_id,
        sum(sr_return_quantity) sr_item_qty
 from store_returns,
      item,
      date_dim
 where sr_item_sk = i_item_sk
 and   d_date    in
(select d_date
from date_dim
where d_week_seq in
(select d_week_seq
from date_dim
  where d_date in ('1999-04-11','1999-08-16','1999-11-11')))
 and   sr_returned_date_sk   = d_date_sk
 group by i_item_id),
 cr_items as
 (select i_item_id item_id,
        sum(cr_return_quantity) cr_item_qty
 from catalog_returns,
      item,
      date_dim
 where cr_item_sk = i_item_sk
 and   d_date    in
(select d_date
from date_dim
where d_week_seq in
(select d_week_seq
from date_dim
  where d_date in ('1999-04-11','1999-08-16','1999-11-11')))
 and   cr_returned_date_sk   = d_date_sk
 group by i_item_id),
 wr_items as
 (select i_item_id item_id,
        sum(wr_return_quantity) wr_item_qty
 from web_returns,
      item,
      date_dim
 where wr_item_sk = i_item_sk
 and   d_date    in
(select d_date
from date_dim
where d_week_seq in
(select d_week_seq
from date_dim
where d_date in ('1999-04-11','1999-08-16','1999-11-11')))
 and   wr_returned_date_sk   = d_date_sk
 group by i_item_id)
```

Figure B.2: Q83

```
 select top 100 sr_items.item_id
      ,sr_item_qty
      ,sr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 sr_dev
      ,cr_item_qty
      ,cr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 cr_dev
      ,wr_item_qty
      ,wr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 wr_dev
      ,(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 average
from sr_items
    ,cr_items
    ,wr_items
where sr_items.item_id=cr_items.item_id
  and sr_items.item_id=wr_items.item_id
order by sr_items.item_id
        ,sr_item_qty
;
```

Figure B.2: (continued)

References

[1] R. Agrawal et al., “The Claremont Report on Database Research.”, *Commun. of ACM*, 2009.

[2] US Environmental Protection Agency, “Report to Congress on Server and Data Center Efficiency”: Public Law 109-431, Aug. 2007.

[3] M. Poess and R. Nambiar., “Energy Cost, The Key Challenge of Today’s Data Centers: A Power Consumption Analysis of TPC-C Results.”, In *Proc. of VLDB* 2008.

[4] Z. Xu, Yi Tu and X. Wang., “Exploring Power-Performance Tradeoffs in Database Systems.”, In *Proc. of ICDE* 2010.

[5] C. Xian, Le Cai and Y. Lu., “Power Measurement of Software Programs on Computers With Multiple I/O Components.”, *IEEE Transactions on Instrumentation and Measurement* 2007 .

[6] Y.-H. Lu, L. Benini and G. D. Micheli, “Operating-system directed power reduction.”, In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED),* 2000.

[7] M. Weiser, B. Welch, A. J. Demers and S. Shenker., “Scheduling for Reduced CPU Energy.”, In *Proc. of the First symposium on Operating Systems Design and Inplementation (OSDI),* 1994.

[8] H. Zeng, C. S. Ellis, A. R. Lebeck and A. Vahdat, “ECOSystem: Managing Energy as a First Class Operating System Resource”, In *Proc. of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS),* 2002.

[9] O. Ozturk and M. Kandemir., “Energy Management in Software-controlled Multi-level Memory Hierarchies”, In *Proc. of GLSVLSI’05,* April 17-19, 2005, Chicago, Illionis, USA.

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

[11] J. Pisharath, A. Choudhari and M. Kandemir., “Reducing Energy Consumption of Queries in Memory Resident Database Systems.”, *Proc. of CASES* 2004.

[12] W. Felter et al., "A Performance-conserving Approach for Reducing Peak Power Consumption in Server Systems", In *Proc. of 19th Intl. Conference on Supercomputing,* 2005

[13] M. Elnozahy, M. Kistler, and R. Rajamony., "Energy-efficient Server Clusters", In *Proc. of the 2nd Workshop on Power-Aware Computing Systems,* 2002.

[14] N. An, S. Gurumurthi, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir and M.J. Irwin., "Energy-performance Trade-offs for Spatial Access Methods on Memory-resident Data.", In *The VLDB Journal,* 11(3):179-197, 2002.

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

[16] SPECpower_ssj2008, `http://www.spec.org/power_ssj2008`.

[17] TPC-Energy Benchmark Development, `http://www.tpc.org/tpc_energy/default.asp`.

[18] S. Rivoire, M.A. Shah, P. Ranganathan and C. Kozyrakis., "JouleSort: A Balanced Energy-efficient Benchmark.", In *Proc. of the ACM International Conference on Management of Data (SIGMOD),* pages 365-376, 2007.

[19] S. Harizopoulos, M.A. Shah, J. Meza and P. Ranganathan., "Energy Efficiency: The New Holy Grail of Data Management Systems Research.", In *Proc. of 4th Biennial Conference on Innovative Data Systems Research (CIDR),* January 4-7, 2009, Asilomar, California, USA.

[20] G. Graefe., "Database Servers Tailored to Improve Energy Efficiency.", In *Proc. of the 2008 EDBT Workshop on Software Engineering For Tailor-Made Data Management* (Nantes, France, March 29-29, 2008).

[21] F. Reiss and T. Kanungo, "A Characterization of the Sensitivity of Query Optimization to Storage Access Cost Parameters", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2003.

[22] A. Hulgeri and S. Sudarshan, "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions", *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2002.

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

[24] N. Reddy and J. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers", *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB)*, August 2005.

[25] Harish D., P. Darera and J. Haritsa, "Identifying Robust Plans through Plan Diagram Reduction", *Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2008.

[26] A. Dey, S. Bhaumik, Harish D. and J. Haritsa, "Efficient Generation of Approximate Plan Diagrams", Tech. Rep. TR-2008-01, DSL/SERC, Indian Inst. of Science, 2008. `http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2008-01.pdf`.

[27] M. Poess, R. O. Nambiar and D. Walrath., "Why You Should Run TPC-DS: A Workload Analysis.", *Proc. of VLDB, 2007.*

[28] PostgreSQL documentation, `http://www.postgresql.org/docs/8.4/static/`.

[29] D. Tsirogiannis, S. Harizopoulos and M. Shah., "Analyzing the Energy Efficiency of a Database Server.", In *Proc. of ACM SIGMOD*, 2010.