Power-performance tradeoff in database systems

Zichen Xu

University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation
http://scholarcommons.usf.edu/etd/95

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Power-Performance Tradeoffs in Database Systems

by

Zichen Xu

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yicheng Tu, Ph.D.
Xiaorui Wang, Ph.D.
Ken Christensen, Ph.D.

Date of Approval:
July 2, 2009

Keywords: Database Management System, Power Modeling, Power Estimation, Energy Concern, Query Optimization, Feedback Control

© Copyright 2009, Zichen Xu
DEDICATION

To my parents Z.T. Xu and F.Y. Wang, my lovely wife J.Y. Li.
ACKNOWLEDGEMENTS

This project is joint work with Dr. Yi-cheng Tu and Dr. Xiaorui Wang. I am extremely appreciate their inspirational guidance through every step of the work. Also, thanks to Mr. Yefu Wang for his help in establishing the experiment environment.
# TABLE OF CONTENTS

LIST OF TABLES ii

LIST OF FIGURES iii

ABSTRACT iv

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 BACKGROUND 5
  2.1 Database Concept and Query Optimization 5
  2.2 Benchmarking Energy Savings 9
  2.3 Related Work in Architecture Level 12
  2.4 Related Work in System Level 14
  2.5 Related Work in Software Level 15

CHAPTER 3 HYPOTHESIS AND MOTIVATING EXAMPLES 18
  3.1 Rationale Behind the Hypothesis 19
  3.2 A Motivating Example 20

CHAPTER 4 METHODOLOGY 24
  4.1 Power Model 25
  4.2 Plan Evaluation Model 27
  4.3 Power Model Calibration 28
  4.4 Workload 29
  4.5 Experimental Design 30
  4.6 Testbed Description 31

CHAPTER 5 EXPERIMENTAL RESULTS 32
  5.1 Main Results (Task 1) 32
  5.2 Query Characterization (Task 2) 37
  5.3 Multiple Workloads (Task 3) 39
  5.4 Discussions 39

CHAPTER 6 CONCLUSIONS AND FUTURE WORK 45

REFERENCES 46
LIST OF TABLES

Table 3.1  Peak power consumption of various AMD CPUs. 18
Table 3.2  Power consumption of various Seagate Momentus hard disks. 19
Table 3.3  Power costs of basic database operations in the experimental system. 23
Table 4.1  Hardware power specifications in the experimental database server. 26
Table 4.2  Power cost functions for accessing single relations and join operations. 27
Table 4.3  Power costs of three single-table scan operations in model verification experiments. 29
Table 5.1  Performance and power/energy consumption of the experiments using TPC-H workload and TPC-C workload. 36
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure 3.1</th>
<th>A sample query from TPC-H.</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3.2</td>
<td>The estimated processing time and power cost of plans evaluated by the modified PostgreSQL query optimizer for the query shown in figure 3.1.</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>A partial plan tree of the query in figure 3.2 generated by the modified query optimizer.</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Organization of the experiment platform.</td>
<td>25</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Power consumption of the TPC-H workloads under three different database sizes.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Power consumption of the TPC-C workloads under three different database sizes.</td>
<td>34</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Power and processing time of the chosen plans for the 19 queries in the TPC-H benchmark as estimated by the query optimizer.</td>
<td>41</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Power, processing time, and energy consumption of the 19 queries in the TPC-H benchmark as measured by power meter.</td>
<td>42</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Relative quantity of power.</td>
<td>43</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Absolute quantity of power.</td>
<td>44</td>
</tr>
</tbody>
</table>
Power-Performance Tradeoffs in Database Systems

Zichen Xu

ABSTRACT

With the total energy consumption of computing systems increasing at a steep rate, much attention had been paid to the design of energy-efficient computing systems and applications. So far, database system design has focused on improving the performance of query processing. The objective of this study is to explore the potential of energy conservation in relational database management systems. The hypothesis is: by modifying the query optimizer in a Database management system (DBMS) to take the energy cost of query plans into consideration, we will be able to reduce the energy usage of database servers and control the tradeoffs between energy consumption and system performance. In this thesis, we provide an in-depth anatomy of typical queries in various benchmarks and qualitatively analyze the energy profile of such queries. The results of extensive experiments show that power savings in the range of 11% to 22% can be achieved by equipping the DBMS with a simple query optimizer that selects query plans based on both estimated processing time and energy requirements. We advocate more research efforts be invested into the design and evaluation of power-aware DBMSs in hope to reach higher level of energy efficiency.
CHAPTER 1
INTRODUCTION

The main objective of computing system design has been improving performance. With the significant increase in energy consumption of computers, power management has become a critical issue in system design and implementation [45]. Various sources [32, 28, 29] suggest an annual electricity bill of as large as 14 billion US dollars for powering the servers in the United States. In a typical data center, electricity consumed by servers and cooling systems (needed to remove the heat generated by servers) contributes to around 20% of the total ownership cost, equivalent to one-third of the total maintenance cost [15]. This makes energy the second largest item in a typical IT department’s monthly bill (while labour being the number one). It is generally believed that energy cost will continue to increase in the next few years [15, 28, 29, 32].

From an environmental point of view, computing systems contribute to 2% of the world’s carbon footprint [43]. Data centers alone contributes to 0.5% (1.2% in U.S. [32]), with a projected four-fold increase by 2020.

While the above costs are calculated directly from energy consumption, power (i.e., energy consumption per unit time) savings are of more practical importance than energy savings in system design. Power capping has become a serious challenge for data centers in recent years. Controlling power consumption is an essential way to avoid system failures caused by power capacity overload or overheating due to increasing high server density (e.g., blade servers) [27, 55]. Modern high-density servers face an increasing probability of thermal failures due to their continuously decreasing size and increasing demand for computational capabilities. For example, recent studies show that 50% of all electronics failures are related to overheating [58]. In addition, an approximately 15°C increase in temperature could double the failure rate of a disk drive [4]. Therefore, it is important to reduce 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 and thus cooling costs [47]. This can contribute to significant energy savings in the long run as the cooling system could account for up to 50% of total energy consumption of a data center [54, 32].

Due to the above facts, power conservation in computing systems has attracted much attention from government agencies, industrial sectors, and research communities. Standards and benchmarks are quickly adapting to the consideration of power consumption [46, 1, 50]. In the research part, early innovations on power-aware computer servers have concentrated on hardware and system software such as compilers and operating systems [35, 59, 41, 20, 10]. Those build a foundation for a computing environment where: 1) hardware can operate on various power-saving modes in which different power/performance tradeoffs can be achieved; 2) system software can control the operating modes of hardware and reschedule resource requests from applications to save energy. One common theme of system-level research on power-aware computing is that they treat the high-level applications as passive resource consumers (abstracted as workload) that are not aware of the low-level efforts of power reduction. In the past few years, the research community has shifted much interest to power-aware applications [9, 57, 13], which provide significant synergistic values to current research on the hardware and system levels. Our research falls into this category.

In this thesis, we study power consumption patterns and identify power-saving opportunities in database management systems (DBMS). Our ultimate goal is to build power-aware DBMSs that can achieve significant cost savings while maintaining reasonable performance in query processing. Such vision is motivated by the following observations.

First, power reduction in DBMSs is of high economical significance: 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 DBMS the largest consumers of power in all the software applications deployed. It is generally true that the processing capacity of the back-end database servers determines that of the front-end web and application servers. In [48], a back-end to front-end power consumption ratio of 11 : 9.9 is reported. Most database servers are configured with capacity sufficient to handle peak load. This implies opportunities for power savings.
Second, a DBMS possesses salient features that can facilitate effective power control, making it a desirable platform for validating research ideas in general application-level power conservation. Specifically, a DBMS maintains extensive statistics about database states for the purpose of efficient query processing. This information can be directly used to derive the power profile of the database workload and help power-related decision making. Moreover, it can adjust its behavior (e.g., selecting query execution plan, materializing views, building indexes) towards a performance goal. Its query optimizer can generate a large number of execution plans for the same query, allowing various tradeoffs between performance and power cost. Last but not least, it is much like an OS itself by requesting large chunks of resources (e.g., memory buffer and disk space) from the OS and managing the distribution of such resources among its users. This provides opportunities to optimize resource management towards high power-efficiency inside the DBMS. It is also for the above reasons that power-aware DBMS can be complementary to OS-level power management solutions and yield greater power savings.

The main objective of this study is to validate our vision on power-aware DBMSs (PDBMS) and provide solutions to some of the main problems related to building such systems. Specifically, we want to answer the following questions:

- Are there really opportunities to save energy with PDBMS? In other words, are existing DBMSs inefficient in terms of power consumption?
- If so, what are the reasons for such inefficiency?
- What can we do in PDBMSs to remedy the above problems? and
- How much power do we expect to save in a PDBMS?

This thesis reports empirical results to address the above issues. Specifically, we profile the power consumption patterns of individual queries in popular TPC benchmarks by both modeling-based and experimental approaches. From these profiles, we identify significant potential for saving power and performing power-performance tradeoffs. We also propose a strategy to make DBMSs power-efficient: redesign the query optimizer to take the power cost of query plans into consideration. Via extensive experiments under different workloads, we show that a DBMS equipped with such a query optimizer can achieve up to 22% reduction of active
power. Note that the main goal of this study is to validate this strategy by an initial design of the optimizer and we leave more sophisticated PDBMS design and implementation as future work.

This thesis states that it is possible to save power, or even energy in manipulating optimizer in database system with full support of experimental result. In another word, there is potential that manipulating software itself to reach a promising saving in power. Also, it provides a free scratch skeleton of testing a P-DBMS with verification of its power-estimation data model. At last, the thesis points out several possible directions of power saving in DBMS in further research.

The thesis is organized as follows: related work summarized in chapter 2; our hypothesis is stated and so are motivating examples in chapter 3; chapter 4 describes the methodology and environment for our experiments; we present and interpret the experimental results in chapter 5 and conclude this thesis with discussions on future research plans in chapter 6.
CHAPTER 2

BACKGROUND

2.1 Database Concept and Query Optimization

A database is a structured records or data collection. It stores and maintains itself in a computer system. According to the specific data model, database organize the data. With different models, various structures are established, such as System R[5]. System R gives the model that is most commonly used today, which is called the relational model. Other models such as the hierarchical model [31] and the network model [6] use a more explicit representation of relationships.

Depending on the intended purpose, there are a number of database architectures in use. Many databases use a combination of strategies. There are two main basis DBMS architectures. One is on-line Transaction Processing systems (OLTP) often use a row-oriented data store architecture, while data-warehouse and other retrieval-focused applications like Google’s BigTable [19], or bibliographic database (library catalogue) systems may use a Column-oriented[53] DBMS architecture.

Document-Oriented, knowledge bases, XML, as well as frame databases and RDF-stores (as known as triple-stores), also use a combination of these architectures in their implementation. It should be noted that not all databases have or need a database ‘schema’ (so called schema-less databases). Over many years the database industry has been dominated by General Purpose database systems, which offer a wide range of functions that are applicable to many, if not most circumstances in modern data processing. These have been enhanced with extensible data types, pioneered in the PostgreSQL project\(^1\), to allow a very wide range of applications to be developed. In our experiments, PostgreSQL is used as our test bench to certify our hypothesis, verify our power/energy model and implement it.

\(^1\)http://www.postgresql.org/
In all of these databases, they can take advantage of indexing[22] to increase their performance. This technology has advanced tremendously since its early uses in the 1960s and 1970s. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Typically, indexes are also stored in the various forms of data-structure mentioned above (such as B-trees, hashes, and linked lists). Usually, a specific technique is chosen by the database designer to increase efficiency in the particular case of the type of index required.

Most relational DBMS’s and some object DBMSs have the advantage that indexes can be created or dropped without changing existing applications making use of it. The database chooses between many different strategies based on which one it estimates will run the fastest. In other words, indexes are transparent to the application or end-user querying the database; while they affect performance, any SQL command will run with or without index to compute the result of an SQL statement. The rational DBMS (RDBMS) will produce a plan of how to execute the query, which is generated by analyzing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are nested loop join, sort-merge join and hash join. Which of these is chosen depends on whether an index exists, what type it is, and its cardinality.

Query optimization² is a function of many RDBMSs in which multiple query plans for satisfying a query are examined and a good query plan is identified as discussed in the index description above. There is no one hundred percent guarantee for an absolute best strategy because there are many ways of creating plans. The first trade off is between the amount of time spent figuring out the best plan and the amount running the plan. Different qualities of database management systems have different ways to balance these two. In PostgreSQL, cost based query optimizers evaluate the resource footprint of various query plans and use this as the basis for plan selection.

Typically the resources which are costed are CPU path length (CPU tuples), amount of disk buffer space (used pages) and interconnect usage between units of parallelism (additional cost).

²http://en.wikipedia.org/wiki/Query_optimizer
The backend process of DBMS forms a set of query plans examined by examining possible access paths (e.g., sequential scan, index access scan and bitmap scan) and various relational table join techniques (e.g., merge join, hash join, nest loop join). The search space can become extremely large depending on the complexity of the SQL query because the number of join plans increase exponentially. There are two main types of optimization. These consist of logical optimization which generates a sequence of relational algebra to solve the query. In addition, there is physical optimization which is used to determine the means of carrying out each operation.

The goal of optimization is to eliminate as many unneeded tuples, or rows as possible. The following is a look at relational algebra as it eliminates unneeded tuples. The project operator is straightforward to implement if \( \text{resourse vector} \) in chapter 3 contains a key to relation R. If it does not include a key of R, it must be eliminated. This must be done by sorting (see sort methods below) and eliminating duplicates. This method can also be done by hashing to eliminate duplicates Hash table. Then SQL command distinct is considered; this does not change the actual data. This just eliminates the duplicates from the results. More over, operations sets are collected. Database management heavily relies on the mathematical principles of set theory which is key in comprehending these operations.

Union, intersection and set difference display all that appear in both sets, each listed once. For intersection, it is more restricted that it only lists items whose keys appear in both lists. Nevertheless, set difference lists all items whose keys appear in the first list but not the second one. These three must be union compatible which means all sequences of selected columns must designate the same number of columns. The data types of the corresponding columns must thereby comply with the conditions valid for comparability. Each data type can be compared to itself. For examples, columns of data type CHAR with the different ASCII and EBCDIC code attributes can be compared to each other, whereby they are implicitly adapted; columns with the ASCII code attribute can be compared to date, time, and time stamp specifications; all numbers can be compared to each other. The last is Cartesian Product, such as \( (R \times S) \), which takes a lot of memory because its result contains a record for each combination of records from R and S.
The performance of a query plan is determined largely by the order in which the tables are joined. For example, when joining 3 tables A, B, C of size 10 rows, 10,000 rows, and 1,000,000 rows, respectively, a query plan that joins B and C first can take several orders-of-magnitude more time to execute than one that joins A and C first.

In PostgreSQL, it follows the same mechanism. For each planning problem, therefore, there will be a list of relations that are either base relations or join relations constructed per sub-joins-lists. These relations joined together in any order the planner sees fit. The standard planner does this as follows:

- Consider joining each plan to each other plans for which there is a usable join clause, and generate a cpu path for each possible join method for each such pair. (If there exists a plan with no join clauses, a clauseless Cartesian-product join is generated. As it is said above, Cartesian Product is costly. Thus, joining that relation to each other available relations. But in the presence of join clauses, it will only consider joins that use available join clauses. Note that join-order restrictions induced by outer joins and IN clauses are treated as if they were real join clauses, to ensure that a workable join order is found in cases where those restrictions force a clauseless join to be done.)

- If there are only two relations in the prepare list, the schedule work is done: Pick the cheapest path for the join plan. If there are more than two, as the above example with plans of different rows, it always considers ways of joining join plans to each other to make join plans that represent more than two list items.

- The join tree is constructed using a "dynamic programming" algorithm\(^3\): the first pass (already described) considers ways to create join relations representing exactly two list items. The second pass considers ways to make join relations that represent exactly three list items; the next pass, four items, etc. The last pass considers how to make the final join relation that includes all list items. (obviously there can be only one join relation at this top level, whereas there can be more than one join relation at lower levels. At each level, it joins that follow available join clauses, if possible, just as described for the first level.)

\(^3\)IBM’s System R database project, http://www.mcjones.org/System_R/
In this manner, a query plan is eventually produced that joins all the queries in the relation. Note that the algorithm keeps track of the sort order of the result set produced by a query plan, also called an interesting order. During dynamic programming, one query plan is considered to beat another query plan that produces the same result, only if they produce the same sort order. This is done for two reasons. First, a particular sort order can avoid a redundant sort operation later on in processing the query. Second, a particular sort order can speed up a subsequent join because it clusters the data in a particular way. Details and examples are presented in chapter 3 and fig. 3.3.

2.2 Benchmarking Energy Savings

The Standard Performance Evaluation Corporation (SPEC) is a non-profit organization that aims to produce "fair, impartial and meaningful benchmarks for computers." SPEC was founded in 1988 and their goal is to ensure that the marketplace has a fair and useful set of metrics to differentiate candidate systems. The benchmarks aim to simulate "real-life" situations and problems. SPECweb2005, for example, tests web server performance by performing various types of parallel HTTP requests. The various tasks are assigned weights based on their perceived importance; these weights are used to compute a single benchmark result in the end. It is written in a platform neutral programming language (usually C or Fortran), and the code may be compiled using whatever compiler for preferable platform, but may not change itself. Manufacturers have been known to optimize their compilers to improve performance of the various SPEC benchmarks. SPEC is particularly well-positioned to tackle this challenging task, and right now given the organization’s experience and expertise in system performance measures and the breadth of our existing benchmarks called SPECpower. It is the first industry-standard SPEC benchmark that evaluates the power and performance characteristics of volume server class and multi-node class computers. Most of previous energy work on architecture and system level are using the outputs of this benchmarks to publish their experimental result. However, our current task is not hardware or system related and SPEC is not much a database related benchmark. For all those reasons, SPEC is not taken my current work bench and previous

---

4Official website, http://www.spec.org/
work. But SPEC is placed for my future work experiments. For database related issue, another standard benchmark is taken into consideration.

The Transaction Processing Performance Council (TPC) is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. The term “transaction” is often applied to a wide variety of business and computer functions. Whilst TPC benchmarks certainly involve the measurement and evaluation of computer functions and operations as SPEC, and it involves database which is my target. Thus, this benchmark is used in my previous work regards as benchmark. The TPC regards a transaction as it is commonly understood in the business world: a commercial exchange of goods, services, or money. A typical transaction, as defined by the TPC, would include the updating to a database system for such things as inventory control (goods), airline reservations (services), or banking (money). In these environments, a number of virtual “customers” or “service” representatives input and manage their transactions via a terminal or desktop computer connected to a database. Typically, the TPC produces benchmarks that measure transaction processing (TP) and database (DB) performance in terms of how many transactions a given system and database can perform per unit of time, e.g., transactions per second or transactions per minute. In this kind of simulation, there are perfectly different kinds of queries and request which will consumed in large database system. With those information, the hypothesis in chapter 3 can be verified.

More specifically, TPC is about to approve the third in its series of benchmarks which measure the performance and price/performance of transaction processing systems. For example, the new TPC Benchmark C, or TPC-C, is an on-line transaction processing (OLTP) benchmark. TPC-C contains multiple transaction types, more complex database, and overall execution structure. TPC-C is based on a workload presented to the TPC two years ago and refined representing a cross section of the industry.

The goal of TPC-C benchmarks is to define a set of functional requirements that can be run on any transaction processing system, regardless of hardware or operating system. It is then up to the test sponsor to submit proof that they have met all the requirements. This

---

5Official website, http://www.tpc.org/
methodology allows any vendor, using “proprietary” or “open” systems, to implement the TPC benchmark and guarantees to end-users that they will see an apples-to-apples comparison. This is a dramatic departure from most other benchmarks where test sponsors are limited to comparing machines that run on just one operating system or benchmarks that execute the same set of software instructions.

TPC benchmarks also differ from other benchmarks in that TPC benchmarks are modeled after actual production applications and environments rather than stand-alone computer tests which may not evaluate key performance factors like user interface, communications, disk I/Os, data storage, and backup and recovery. As an OLTP system benchmark, TPC-C simulates a complete environment where a population of terminal operators executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. However, it should be stressed that it is not the intent of TPC-C to specify how to best implement an Order-Entry system. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but, rather, represents any industry that must manage, sell, or distribute a product or service.

In the TPC-C business model, a wholesale parts supplier (called the Company below) operates out of a number of warehouses and their associated sales districts. The TPC benchmark is designed to scale just as the Company expands and new warehouses are created. However, certain consistent requirements must be maintained as the benchmark is scaled. Each warehouse in the TPC-C model must supply ten sales districts, and each district serves three thousand customers. An operator from a sales district can select, at any time, one of the five operations or transactions offered by the Company’s order-entry system. Like the transactions themselves, the frequency of the individual transactions are modeled after realistic scenarios.

The most frequent transaction consists of entering a new order which, on average, is comprised of ten different items. Each warehouse tries to maintain stock for the 100,000 items in the Company’s catalog and fill orders from that stock. However, in reality, one warehouse will probably not have all the parts required to fill every order. Therefore, TPC-C requires
that close to ten percent of all orders must be supplied by another warehouse of the Company. Another frequent transaction consists in recording a payment received from a customer. Less frequently, operators will request the status of a previously placed order, process a batch of ten orders for delivery, or query the system for potential supply shortages by examining the level of stock at the local warehouse. A total of five types of transactions, then, are used to model this business activity. The performance metric reported by TPC-C measures the number of orders that can be fully processed per minute. Although the TPC-C specifications are public, there was not a public implementation: each one has to develop and use its own.

The TPC Benchmark H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric, and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. More details about how TPC benchmarks involve in the test are in chapter 4.

2.3 Related Work in Architecture Level

Jumping into energy concern topics in computer science level, the first amount work comes from hardware and architecture. [23] states energy dissipation and puts an investigation about how super-scalar and pipelining issues affect the energy-delay product of general purpose processors. It states clearly that idealized machines pipelining would reach approximately a two-times improvement in energy-delay product. Nevertheless, super-scalar issue only improves the energy-delay product by a small amount. Thus, improving the efficiency will be difficult since the overhead is distributed among many units, none of which represent a significant fraction of the energy dissipation.
Other than this work, [36, 25, 21] presents its extension in embedded system, prototyping framework and asynchronous microprocessors. With constraint of low energy dissipation, [36] works out an typical indispensable peculiarity of embedded mobile computing systems and also the first comprehensive framework that simultaneously evaluates the tradeoffs of energy dissipations of software and hardware such as caches and main memory. Another prototyping framework in [25] addresses the issues of power optimization and estimation for hardware and software embedded systems. It is based on a generic embedded system architecture consisting of embedded CPU, custom hardware, and memory hierarchy. More over, it rapidly evaluate/estimate power and performance and thus facilitate comprehensive design space explorations. Another extraordinary result comes as power reductions of up to 94% without performance losses and only a slight increases in total chip size (i.e., transistor count). [21] also presents an asynchronous microprocessors called Amulet implementations of the ARM 32-bit RISC architecture. Their asynchronous control framework has positive benefits for low-power applications because it reduces activity to the minimum required to perform a task, whereas a clock inevitably incurs wasteful activity.

Digging deeper in microarchitecture frame work, [11] certify the ability to estimate power consumption in an early-stage definition. There exists trade-off studies which is a key new methodology enhancement. As it is stated, “saving power can be exposed via microarchitecture-level modeling, particularly through clock-gating and dynamic adaptation.” In the work of [30], it not only gives a scaling power-aware microarchitecture and presents a strategy for run-time profiling to optimize the configuration of a microprocessor dynamically. In the way as describe, saving power with minimum performance penalty is achieved. After changes in the configuration of the processor according to the parallelism in the running program, the work observed a decrease of up to 23% in energy/cycle and up to 8% in energy per instruction. Rather than the strategy and frame work, they do not focus thermal issue, another work [52] comes out. Obviously, designing cooling solutions for worst-case power dissipation is prohibitively expensive. Chips that can autonomously modify their execution and power-dissipation characteristics permit the use of lower-cost cooling solutions while still guaranteeing safe temperature regulation. It introduces several effective ways for dynamic thermal management (DTM): “temperature-
tracking” frequency scaling, “migrating computation” to spare hardware units, and a “hybrid” policy that combines fetch gating with dynamic voltage scaling. By exploiting instruction-level parallelism, the latter two achieve their performance advantage. Vice versa, it shows the importance of microarchitecture research in helping control the growth of cooling costs. Rather this work on thermal issues in microarchitecture, [37] studies microarchitecture-level temperature and voltage aware performance and power modeling. The modeling temperature work at the microarchitecture level also shows that power metrics are poor predictors of temperature ranging from various $V_{dd}$, that sensor imprecision has a substantial impact on the performance of DTM, and that the inclusion of lateral resistances for thermal diffusion is important for accuracy. It points out voltage scaling the same as frame scaling in [30].

2.4 Related Work in System Level

System-level power-related research has created an ocean of literature. A very first basic tutorial [7] surveys design methods for energy-efficient system-level design. The testbed electronic systems consists of a hardware platform and software layers. The three major constituents of hardware that consume energy, namely computation, communication, and storage units, and methods of reducing their energy consumption are reviewed and so are the models for analyzing the energy cost of software, methods for energy-efficient software design and compilation. From this survey, energy-efficient design for conceptualization and modeling design and implementation, and runtime management are stated so that a bird view of system level power efficient works is shown at very first stage.

To implement such technique, scheduling work comes as the first work field that researchers focuses on. Especially in the real time embedded system who concerns performance and power as both critical mission. In [39], a new scheduling technique for supporting the design and evaluation of a class of power-aware systems in mission-critical applications is established. It satisfies constraints of stringent min/max timing and max power and also makes the best effort to satisfy the min power constraint in an attempt to fully utilize free power or to control power jitter. Another work on the similar topic is in [38], it states that “power-aware systems are not just low-power but must track their power sources and the changing power and performance
constraints imposed by the environment. Moreover, they must fully explore and integrate many novel power management techniques.” Thus, it proposes a new graph-based model to integrate novel power management techniques and facilitate design-space exploration of power-aware embedded systems. With capturing min/max timing and min/max power constraints on computation and non-computation tasks through a new constraint classification, it enables derivation of flexible system-level schedules.

Some analysis work on the power consumption of real-time operating systems (RTOSs) is done in [16], which is an important component of the system software layer. This work is also a basis of many other papers cited in section 2.5. This work presents the power profiles for a commercial RTOS running several applications on an embedded system based on the Fujitsu SPARC Lite processor. Demonstration about the RTOS can consume a significant fraction of the system power and, in addition, impact the power consumed by other software components.

As mentioned earlier, much of the prior work has either attempted to reduce power consumption by improving the power-efficiency of individual hardware components [35], or focused on system-level power and thermal management [10, 20, 24, 41, 59, 49, 56]. The common philosophy behind these is: power consumption should be proportional to the load put on the system. In order to save energy, we need to turn the hardware to power-saving modes when the workload on the system is light. Among these work, Zeng et al. [59] manages power in operating system which provides us with an initial power model for power cost estimation of query plans. In this thesis, we will show that, given the same load of user queries, the DBMS can choose to handle the load in a power-saving manner.

2.5 Related Work in Software Level

Application-level power-aware computing has become an active research field for the following reasons: 1) Low-level power optimization depends heavily on accurate estimation of the power profiles of applications. However, much information and statistics needed for such estimations are only available inside the applications. Workload abstraction is often found to be an oversimplified description of application behaviors; 2) Many applications have different execution paths to accomplish the same computational task. Power-aware applications, which
adaptively adjust its behaviors according to the power-related states of underlying systems, can provide additional opportunities for power saving. Many research projects in this area are on web services \[9, 12, 14, 17, 26, 51, 60\]. Recent target application include MapReduce \[13\]. In \[57\], a general-purpose programming framework was developed to allow applications to be executed with different plans according to their power costs.

The power consumption in databases has just started drawing attention from the research community. The Claremont report on database research \[2\] explicitly states the importance of “designing power-aware DBMSs that limit energy costs without sacrificing scalability ...”. Two recent projects address power consumption issues in data processing applications and data centers: \[50\] presents a benchmark for evaluating the energy efficiency of implementations of various sorting algorithms. Due to the large volume of data, such algorithms are I/O intensive and therefore the I/O access patterns have profound effects on the energy efficiency of the algorithms. \[48\] reports extensive experimental results on the power consumption patterns in typical commercial database servers. Based on these results, it provides suggestions on how to make the system more power efficient. However, these suggestions focus on utilizing new hardware rather than modifying the DBMS software. To the best of our knowledge, our project is the first one that takes the path of redesigning the DBMS kernel for power-saving purposes.

A very similar work as our current database energy saving job is \[42\]. It focuses on software level application to control power management. Two algorithms of dynamic power management implement for controlling the power states of a hard disk. It creates a template for software-controlled power management and experimental comparisons of management algorithms for a hard disk.

While limiting power consumption is obviously necessary in mobile computing systems (e.g., PDAs, laptops) in which battery power is the most stringent resource. The software control issues as follows: transition, load-change, and adaptation produces both software and hardware problems. In \[40\], it implements and proposes solutions to software energy management issues created by existing and suggested hardware innovations.

In all, as stated in \[33\], “energy-proportional designs would enable large energy savings in servers, potentially doubling their efficiency in real-life use.” Using energy usage profile of every
system component, will contribute to a significant improvement in saving energy. Our work in this thesis, is mainly focus on opportunities in large servers with unique purpose, database computing since currently there are very few efforts [3] devoted to power-aware database systems on servers connected to power grids. But this topic is increasingly attractive to database researchers now.
CHAPTER 3

HYPOTHESIS AND MOTIVATING EXAMPLES

In response to the questions list in Section 1 (especially questions 1 and 3), our main hypothesis in this study is: There exist power saving opportunities in current DBMSs, and we can grab those opportunities by designing a query optimizer that takes power cost into consideration in query evaluation.

The above hypothesis is intuitive due to the well-known fact that the current query optimization mechanisms consider performance as the sole goal. Therefore, the lack of power considerations makes them unsuitable for saving power. On the other hand, it could also be counterintuitive to many since it is possible that query optimization towards shortest processing time coincides with that towards lowest power cost, and therefore no power savings can be achieved by modifying the behavior of the database engine. In fact, this has been a frequent argument we encounter in our communications with fellow researchers. The argument is based on the understanding of query processing time being a measure of some “load” to be finished by the DBMS, and more “load” the system faces, more energy will be consumed. Most of the power-aware computing research follow this line of reasoning. However, more careful analysis of the different types of resources associated with such “load” to be handled in the DBMS will lead to a different conclusion.

<table>
<thead>
<tr>
<th>Processor</th>
<th>TDP(W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athlon 64 3500+</td>
<td>67</td>
</tr>
<tr>
<td>Athlon 64 3700+</td>
<td>89</td>
</tr>
<tr>
<td>Athlon 64 FX-55/57</td>
<td>104</td>
</tr>
<tr>
<td>Opteron 2.8GHz</td>
<td>93</td>
</tr>
<tr>
<td>Opteron Dual Core 2.2GHz S</td>
<td>93</td>
</tr>
<tr>
<td>Opteron Dual Core 1MB Cache 64 2.4GHz</td>
<td>95</td>
</tr>
<tr>
<td>Opteron Dual Core 1MB Cache 64 3GHz</td>
<td>98</td>
</tr>
</tbody>
</table>
Table 3.2. Power consumption of various Seagate Momentus hard disks.

<table>
<thead>
<tr>
<th>Disk</th>
<th>Peak Power(W)</th>
<th>Idle Power(W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5400.5, 320GB, SATA</td>
<td>2.45</td>
<td>0.92</td>
</tr>
<tr>
<td>7200.5, 320GB, SATA</td>
<td>3.03</td>
<td>0.95</td>
</tr>
<tr>
<td>7200.5, 100GB, ATA</td>
<td>3.03</td>
<td>0.95</td>
</tr>
<tr>
<td>7200.5, 160GB, SATA</td>
<td>3.83</td>
<td>1.10</td>
</tr>
<tr>
<td>7200.5, 100GB, SATA</td>
<td>4.03</td>
<td>1.74</td>
</tr>
</tbody>
</table>

3.1 Rationale Behind the Hypothesis

In traditional performance-driven query optimizers, the plan cost is the summation of the holding times of all system resources including CPU, disks, and communication channels (in case of distributed databases). However, the increase of the processing speed of modern CPUs outpaces that of the I/O speed by orders of magnitude, making CPU time negligible as compared to I/O time in most cases. Therefore, query optimization becomes essentially a problem of finding the plan with the smallest number of I/O operations instead of “total” processing time.\(^1\) In the paradigm of energy-aware computing, however, the weight of CPUs in terms of power consumption is greater than (or at least comparable to) that of the storage systems [48]. For example, according to Tables 3.1\(^2\) and 3.2\(^3\) (in Table 3.1, TDP stands for thermal design power), power consumption of one Opteron Dual Core 3GHz CPU equals that of 24 SATA disks with 100GB volume each. Given this, we can easily see that the existing query optimization mechanisms in DBMSs are ill-suited for saving power. Consider the following scenario: for two execution plans A and B of the same query, if the I/O cost of plan A is slightly higher than that of plan B but it uses much less CPU time than B, the current query optimizer will choose B since I/O is always the dominant factor in total processing time. However, a query optimizer that considers both power and performance could choose A instead. This is because plan B’s high CPU requirement translates into a (much) higher power consumption while it is only marginally better than A in (estimated) processing time. In this study, we explored the power and performance costs of a large number of plans for the TPC-H queries and found many

\(^1\)In the case of distributed databases, communication bandwidth plays such a dominant role (or at least it is as important as the I/O time). However, we focus on a single server setup in this thesis and leave the study of distributed databases as future work.

\(^2\)Data source: http://www.amd.com/, TDP means thermal design power

\(^3\)Data source: http://www.seagate.com/
SELECT s_acctbal, s_name, n_name, p_partykey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey
   AND s_suppkey = ps_suppkey
   AND p_size = 15
   AND p_type like '%BRASS%'
   AND s_nationkey = n_nationkey
   AND n_regionkey = r_regionkey
   AND r_name = 'EUROPE'
ORDER BY s_acctbal desc, n_name, s_name, p_partkey
LIMIT 100;

Figure 3.1. A sample query from TPC-H.

cases that match the above scenario. One representative case will be demonstrated in Section 3.2.

To explore the search plans generated by PostgreSQL, we carefully modify its plan generation process such that we can see all possible plans of a query. In the regular PostgreSQL system, a heuristic search algorithm that drops intermediate plans greedily is used, in order to achieve polynomial time traverse of the space that grows exponentially with the number of tables involved. However, it also keeps a trigger to allow exhaustive search, and we pull the trigger to collect data for all the possible plans. Tens of thousands of plans could be generated in the backend PostgreSQL server when a query is in preprocessing stage, along with their estimated time cost and power cost. The latter is rendered by our power model that will be discussed in Section 4.1.

3.2 A Motivating Example

Here we show one example with a query extracted from the TPC-H benchmark.4 As shown in Fig. 3.1, this is a query that retrieves eight attributes with seven conjunctive conditions upon joining five tables. There are about 30000 plans generated before the final execution plan is selected. Most of the 30000 plans are inferior outliers that most likely would have been

4Specifically, it is query No. 2 in the original TPC-H benchmark tool.
dropped in the heuristic search process (if the trigger was not pulled). The estimated costs of some of the more interesting plan nodes are presented in Fig. 3.2. In this graph, if a plan A dominates (i.e., bears lower power and time costs) another plan B, we can be sure that B should not be considered. However, any plan in the non-dominant frontier of the graph can be chosen. Which one to choose will depend on the level of energy-performance tradeoff (e.g., how much performance to sacrifice to save 1% power) we are willing to make (more details can be found in Section 4.2).

From Fig. 3.2, we can see there are at least two plan nodes on the non-dominant frontier, which are shown as circled dots in the magnified portion of the graph. The right-circled node (on the right) has a power cost of 5.12 and processing time of 5.15 while the left node to its left has power and time costs of 4.16 and 5.62, respectively. If we use the original query optimizer, the right one would have been picked because of its small estimated time cost. However, if we choose the left node, although the time may increase to 5.62, the power cost decreases to 4.16 – that is to trade a 5% performance degradation for a 18.75% power saving. This saving is substantial if the estimation comes with reasonable precision. Apparently, such tradeoffs can be utilized to achieve power reduction. Note here that the larger search space for one query is, the more likely such tradeoffs can be found. On the other hand, single table query scheduling has limited power-saving potential in our current method.

In summary, we believe the descriptions in Section 3.1 and the above motivating example provides an answer to question 2 listed in Section 1. We will also verify our answer experimentally in Section 5.2.

In addition to the costs of the above interesting plans, we also explored their computational details for a low-level explanation of the cost differences between them. The main difference between the join paths of these two plans is: the last join in the green node is a hash join while the one for the red node is a merge join. In Fig. 3.3, we present such details in the form of the plan trees generated by the PostgreSQL query optimizer. Each square in the graph represents a partial plan: a single-table scan (on the leaf level) or join of multiple tables (on higher levels). Each square contains the access method (algorithm) name followed by the IDs of tables to be joined, and the estimated energy and time costs of the plan in (x, y)
format. Each plan is generated by joining two lower-level plans. For example, plan M, which
is a hash of tables 1-4, is generated by joining plan J (hash join of tables 1,2,4) and plan C
(sequential scan of table 3). The top plans O and P correspond to the red and green dots
in Fig. 3.2, respectively. The join sequence of plan O is: 1) A nested-loop join B → F; 2) F hash join D → J; 3) J hash join C → M; and 4) M hash join E → O. For plan P: it is: 1) A nested-loop join B → F; 2) F hash join D → J; 3) L hash join D → N; and 4) N hash join C → P.
Figure 3.3. A partial plan tree of the query in figure 3.2 generated by the modified query optimizer.

Table 3.3. Power costs of basic database operations in the experimental system.

<table>
<thead>
<tr>
<th>Function Parameter</th>
<th>Symbol</th>
<th>Default Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_CPU_TUPLE_POWER_COST</td>
<td>α</td>
<td>0.4</td>
<td>Estimated CPU power cost per tuple</td>
</tr>
<tr>
<td>DEFAULT_CPU_INDEX_TUPLE_POWER_COST</td>
<td>τ</td>
<td>0.05</td>
<td>Estimated CPU power cost per indexed tuple</td>
</tr>
<tr>
<td>DEFAULT_PAGE_POWER_COST</td>
<td>β</td>
<td>4.7</td>
<td>Estimated power cost for W/R one page without buffering</td>
</tr>
</tbody>
</table>
CHAPTER 4

METHODOLOGY

To capture the power-saving opportunities in query processing, our main approach is to find query plans with low power cost during query optimization. For this purpose, we modify the current query optimizer to make it incorporate a power model to estimate the power cost of plans, and a query evaluation model that takes both power and performance into account for comparing query plans. We use the open-source PostgreSQL DBMS as our experimental system. We feed the PostgreSQL system with the modified query optimizer with various workloads generated from the TPC-H\(^1\) and TPC-C\(^2\) benchmarks. In this project, we collect two types of experimental data:

- *query processing performance*, which is obtained by recording the starting and ending time of the workload processing period within the DBMS (i.e., ignoring the communication delays); and

- *power consumption*, which is measured by power meters\(^3\) connected to the server via a USB connection. Thus, the power we record is that of the whole server instead of individual hardware components. Although power is the focal point of this thesis, we also discuss energy consumption in several occasions. Note that energy is the product of power consumption and query processing time.

The architecture of our experimental platform is shown in Fig. 4.1. For the ease of reproducing our results, we provide more details of the components in this platform as follows.

\(^1\)http://www.tpc.org/tpch/
\(^2\)http://www.tpc.org/tpcc
\(^3\)Watts up PRO power meter, http://www.wattsupmeters.com
4.1 Power Model

The cost model to estimate the plans’ processing time in PostgreSQL works as follows: it first maps each plan to the number of basic operations (e.g., tuples to process) needed to compute the result following this plan. We call such numbers the operation vector. The plan’s time cost is calculated from a set of static parameters that describe the resource holding time per basic operation (e.g., CPU time to process one tuple). These parameters are different in each machine and are calibrated at compile-time. This part of the query optimizer is not modified in this project.
Table 4.1. Hardware power specifications in the experimental database server.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Memory</th>
<th>Disk</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>98</td>
<td>9 * 4</td>
<td>36</td>
<td>136.63</td>
</tr>
</tbody>
</table>

In this study, we also follow the above strategy to estimate the power cost of plans. We first define a static power profile for each basic operation in query processing and maintain such power costs as system parameters of the DBMS. By this, we can calculate the power cost of an entire plan given its operation vector. The power profile of basic operations for our experimental system is shown in Table 3.3. The initial values of these parameters (data not shown) were obtained from the specifications of the hardware components we find in our server as in 3.1 and 3.2 and divided by related estimated time from PostgreSQL. For example, in a CPU of active power (i.e., peak power minus the idle power) 20 watts, PostgreSQL estimates the time to process one non-indexed tuple to be 0.04ms, the estimated CPU power per tuple would be 20W times 0.04ms and divided by one second – the result is 0.75mW. The power estimation for disk and memory follows the same way.

Moreover, in calculating the power costs, we assume (as did in [48]) “the peak power consumption of an entire system during the measurement interval is identical to the aggregate of the individual nameplate power consumption”. As for main memory, the approximate power consumption is set to 9 watts per DIMM module, as suggested by [18]. The power consumption of disk drives and CPU are obtained from the manufacturers’ web sites. The power specifications of the main hardware components in our experimental system are listed in Table 4.1.

Note that the hardware specifications are provided by the hardware vendors and are best-effort estimations of the real parameters. In order to get more realistic estimations, we run calibration tests to get the final per-operation power costs listed in Table 3.3. More details about model calibration will be presented in 4.3.

The power cost of a plan is calculated from those of the higher-level operations, which consist of basic operations shown in Table 3.3. Table 4.2 lists the formulae for the computation of the power cost of single-relation scans via different access methods and two-table joins. Clearly, these formulae use the values of basic operations as building blocks. Again, we follow the exact same mechanism for calculating time costs in PostgreSQL to generate these formulae.
### Table 4.2. Power cost functions for accessing single relations and join operations.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Cost function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq Scan</td>
<td>$\alpha T + \beta N$</td>
</tr>
<tr>
<td>Index Scan</td>
<td>$\tau T + \beta N + C$</td>
</tr>
<tr>
<td>Bitmap Scan</td>
<td>$n(\tau T + \beta N) + C$</td>
</tr>
<tr>
<td>Subquery Scan</td>
<td>$\alpha T + \beta N$</td>
</tr>
<tr>
<td>Function Scan</td>
<td>$\alpha T + \beta N$</td>
</tr>
<tr>
<td>Nested loop</td>
<td>$P(outer - paths)$</td>
</tr>
<tr>
<td></td>
<td>$\quad + n \times P(inner - paths)$</td>
</tr>
<tr>
<td>Merge</td>
<td>$n \times (P(outer - paths)$</td>
</tr>
<tr>
<td></td>
<td>$\quad + P(inner - paths))$</td>
</tr>
<tr>
<td>Hash</td>
<td>$P(outer - paths)$</td>
</tr>
<tr>
<td></td>
<td>$\quad + n \times P(inner - paths) + C$</td>
</tr>
</tbody>
</table>

#### 4.2 Plan Evaluation Model

The evaluation model is actually a criterion used to evaluate the superiority of alternative query plans towards an optimization goal. The model of current PostgreSQL is simple as it only involves the processing time. In this study, we need a criterion to reflect adjustable tradeoffs between power cost (denoted as $P$) and processing time (denoted as $T$). In general, this cost should a function of $P$, $T$, and $n$, which is the relative weight put on the two former factors. Relate this to Fig. 3.2, all points with the same cost will form a pareto frontier whose shape is determined by the function and parameter $n$. In this thesis, we use a model as suggested by various research projects in hardware [44] and software [34, 50] systems.

Specifically, a metric model of the following format is adopted:

$$C = PT^n = ET^{n-1} \tag{4.1}$$

where $C$ is the aggregated cost and $n$ is a coefficient that reflects the relative importance of $P$ and $T$. Intuitively, it means we are willing to sacrifice a $d^n$-time degradation in performance to achieve a $d^t$-time power reduction. The model is general in that we can be used for different optimization goals with the choice of $n$. When $n = \infty$, we only consider the time cost (i.e., as in the original PostgreSQL does. In practice, we simply disregard $P$ by setting $C = T$.); for $n = 0$, we optimize towards lowest power consumption; and for $n = 1$, power and time performance are both taken into consideration – the cost of a plan is basically its energy consumption.
4.3 Power Model Calibration

An iterative approach is used for calibrating parameters $\alpha, \beta, \tau$. Some assumptions are made in such calibration process. First, the environment, such as temperature, affects the result a lot. It is easy to understand, for an example, the power/energy consumptions of the same set of servers in $10^\circ C$ and $100^\circ C$ will be remarkably different, so are the lives of servers. We assume all the benchmarks are running under the same humidity and temperature. Another important factor in power evaluation is the complexity of other components in the experimental system such as competing applications and system balance [48]. We ignore those factors to simplify the calibration process.

We use workloads consisting of simple queries whose execution plans use only one of the following table scan methods: sequential scan, index scan and bitmap scan. The calibration procedure is as follows:

- Step 1. Select one workload, execute it with the initial $\alpha, \beta$, and $\tau$ values. Record the peak power consumption using a power meter;

- Step 2. Compare the estimated power consumption with the real power cost. If the difference is under 20%,\(^4\) then keep the current values of $\alpha, \beta$, and $\tau$. Otherwise, update the three parameters using the real power consumption, number of tuples processed, and number of pages read. The latter two can be obtained from the calibration workload and the new parameters can be calculated from equations listed in Table 4.2;

- Step 3. Repeat Step 2 by running a different workload; After the test is repeated 1000 times, we terminate the calibration process and the set of parameters obtained will be used as the calibrated values.\(^5\)

We also verified the calibrated parameters by using them to estimate the power costs of high-level operations. Table 4.3 shows the estimated power consumption of the three single-table scan operations calculated from the refined parameters after calibration (in Table 3.3),

\(^4\) The 20% threshold used throughout the calibration procedure is borrowed from [48]
\(^5\) In the calibration procedure, the workloads are generated by varying the percentage of queries using three different scan methods (sequential, index and bitmap). Therefore, there could be a large number of workloads to choose from.
Table 4.3. Power costs of three single-table scan operations in model verification experiments.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Estimated Power (w)</th>
<th>Measure Power (w)</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq Scan</td>
<td>24.1</td>
<td>20.2</td>
<td>7.2%</td>
</tr>
<tr>
<td>Index Scan</td>
<td>35.2</td>
<td>30.8</td>
<td>14.5%</td>
</tr>
<tr>
<td>Bitmap Scan</td>
<td>30.5</td>
<td>28.1</td>
<td>8.5%</td>
</tr>
</tbody>
</table>

compared with those measured by power meter in a scan-only query processing experiment. All the queries in these experiments are executed by a single type of basic table scan methods, without any other scan methods or join operations because it is difficult to tell the power consumption of different operations in a complex query plan. For each experiment involving one basic scan method, we feed the PostgreSQL with 100 such concurrent queries and record the power consumption of the whole server. The estimated values are calculated from formulae listed in Table 4.2 with the $T$ and $N$ valued obtained from the PostgreSQL optimizer. Table 4.3 shows the difference between power model estimation using the calibrated parameters and actual peak power when executing these simple queries. It is clear that the power model overestimates the total power consumption in the server for single plan processing. However, these estimations are considered close enough to the real values. The difference varies from 7.2% to 14.5% – these are very promising results as compared to 18% reported in [48].

4.4 Workload

TPC-C and TPC-H are two mainstream benchmarks for testing database performance. They provide constraints and specifications of various simulations of different real-world tasks, such as business oriented ad-hoc queries and concurrent data modifications. These are the areas where large-scale database servers are heavily used therefore match our expectations of a target database system.

In the experiment, we chose TPC-H as our main testing benchmark. The TPC-H benchmark illustrates decision support systems that examine large volumes of data and executes 22 different types of queries with a high degree of complexity. Two salient features of the TPC-H benchmark

---

*Protocols of such experiments are designed to be very similar to those of our main experiments whose details can be found in chapters 4.5 and 5.*
are: 1) there are official benchmark tools available - these can save us more time in generating workloads following the TPC-H specification; 2) The individual components of the benchmark - the queries - are directly accessible. This provides a chance to study the resource consumption pattern of each query. By this, we can generate different workloads dynamically from query groups that hold common features. For TPC-C, we use an open source implementation named TPCC-UVa,\textsuperscript{7} which implements all the specifications and requirements that are listed in the TPC-C description. We use TPC-C for supplementary tests because, unlike TPC-H, all the queries in TPCC-UVa are not directly accessible.

4.5 Experimental Design

To test our hypothesis and provide further answers to questions listed in Section 1, we perform experiments that can be divided into the following three components:

- Task 1. Test the performance and power consumption of query processing under standard query workloads provided by the TPC benchmarks to explore the potential of power saving in databases. These tests are performed under different database sizes;

- Task 2. Characterize the power profile of individual queries in the TPC-H benchmark for the purpose of identifying sources of power savings (if verified by Task 1) on single-query level. This is expected to build the foundation of synthetic workloads for further exploration on power-aware DBMS;

- Task 3. To further explore the power and energy saving potential in DBMSs, we create workloads with different power consumption patterns and test the impact of such patterns on system performance and power/energy costs. The synthetic workload will be generated based on the characterization work in Task 2.

The detailed protocols for the above tasks will be described when we present the experimental results in Section 5.

We run all our experiments in a single database server equipped with one Dual-core AMD Opteron Processor (type 2222 SE), 4 GB of main memory, and a single 135 GB hard disk. The

\textsuperscript{7} http://www.infor.uva.es/~diego/tpcc-uva.html
DBMS is PostgreSQL release 8.3.3 running on Linux kernel 2.6.14. In all experiments, PostgreSQL is configured to perform heuristic (i.e., non-exhaustive) search in query optimization. Every query, when issued, will be eventually executed until completion. HDD read pre-fetching is disabled and there are no other CPU or I/O intensive applications running on the system. At runtime, the instantaneous power consumption of the whole server is recorded using the Watts up PRO power meter at its highest sampling frequency – one hertz. In the next chapter, we will show that this frequency is sufficient.

4.6 Testbed Description

The testing environment of the current experiments are mainly two sets of servers, details of them (hardware and software) are shown as follows:

- CPU: AMD Dual Core 3333SE 2.40GHz and Intel(R) Core Quad CPU Q6600 2.40GHz
- Cache: Both 4096KB
- Memory: Both 4GB
- Hard Drive: 200 Gigabytes and 480 Gigabytes
- Operating System: Linux Kernel 2.6.14(kernel modified)
- File System: Both Ext2 format
- Database System: PostgreSQL 8.3.3(backend modified)
- Benchmark: TPCC-UVa, TPC-H (latest version)

The kernel of operating system is modified to eliminate the effect of buffer read and buffer write in page switching and file system backup. In this way, it will reduce the effect of power consumption of those kinds of operations. Also, the backend of PostgreSQL is carefully reprogrammed, recompiled and rebuilt to fulfill three purposes: 1. implementing all the data models described in section 4.2 and 4.1; 2. increasing the capacity of the server end of PostgreSQL so that it can process more clients request or halt them simultaneously; 3. reconfigure itself to fit the modified operating system platform.
CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter, we present the results of the three tasks. In all discussions related to power consumption, only the power used for processing the workload will be presented. We call this the active power, which is basically the measured power during query processing less the idle power of the server.

5.1 Main Results (Task 1)

In this section, we present the results of a series of experiments running the standard workload provided by both TPC-H and TPC-C benchmarks. Specifically, we set up 100 clients (i.e., the maximum number of clients allowed in PostgreSQL) to send out queries concurrently. For each client, queries are randomly drawn from the pool of all TPC-H queries. All the clients will finish sending jobs within an one-second window, which means the database server will soon reach its maximum utilization and keep working on full throttle afterwards. We configure the DBMS to the plan evaluation model in Eq. (4.1) with a different time-invariant \( n \) value. Specifically, we choose: 1) \( n = \infty \), which is basically the performance-only scheme in the original PostgreSQL; 2) \( n = 1 \), which reflects equal preference on power and performance; and 3) \( n = 0 \), power-only optimization. We repeat the same experiments under three different database sizes: 500MB, 1GB, and 10GB for TPC-H; 1GB, 5GB, and 10GB for TPC-C.

In Fig. 5.1 and 5.2, we plot the instantaneous power consumption of the workload in the first 800 seconds of each experiment.\(^1\) We can clearly see that, system runs on significantly lower power when we choose a query evaluation model that favors low-power plans. When we compare the energy-only (\( n = 0 \)) with the performance-only (\( n = \infty \)) results, we observe a

\(^1\) More data is available but not plotted here. However, they do not provide more information since, in all experiments, power stabilizes after the first 800 seconds. The total workload processing time can be found in Table 5.1.
stable difference of power with magnitude of 7 to 9 watts, which corresponds to power savings ranging from 11% to 22% (Table 5.1). Without a surprise, the savings of the $n = 1$ case are smaller than those created by the power-only experiment, but it still achieves 11 to 16% power saving. The experiments using the TPC-C workload show comparable levels of power saving.

The power values reported in Table 5.1 are the average power over the entire workload processing period. One might argue that this is not accurate because we may have missed power spikes under the one-second sampling period. However, by reading the data, we believe this should not be a concern. Note that, in all cases, the measured power level is very stable over time without any significant fluctuations. We could have missed some power spikes, but it is very unlikely to miss all of them. In fact, this is also the result of our experimental design: the
workload contains heterogeneous queries sent from 100 clients, the DBMS is able to schedule the resources among these concurrent query sessions, giving rise to smooth resource usage at runtime.

Another side of the story is the performance degradation of power-aware DBMSs. In Table 5.1, we show the total query processing time of the TPC-H experiments. It is obvious that it took more time for the power-aware systems to finish processing all the queries, but the differences are small as compared to the differences of power. As a result, we still yield total energy savings of 3 to 7%.

Figure 5.2. Power consumption of the TPC-C workloads under three different database sizes.
It is interesting that, in the experiment under the 500MB database (Fig. 5.1a), we see a sudden drop of power consumption at the 160th second. This is because the TPC-H workload consists of queries that are bound by I/O and those by computation. All queries in the CPU-bound category finished at around the 160th second therefore the system only runs the I/O bound queries afterwards. The fact that power drops abruptly after the 160th second supports our claim that power consumption of CPUs is significant as compared to that of the disks. From the time when the power drops, we can also see that the processing time of the CPU-bound queries in the power-aware systems are very close to that in the original DBMS. A similar drop can be observed in Fig. 5.1b, Fig. 5.2a, and Fig. 5.2b. At the 200th second in Fig. 5.1b, we can see a sudden jump of power. We believe this is due to unexpected operating system activities.

An interesting result can be seen from the TPC-C experiment with the 5GB database: the actual query processing time under the power-aware query optimizer (120 mins) is even lower than that under the performance-driven query optimizer (123 mins). This is due to the well-known fact that PostgreSQL does not guarantee the selection of the optimal plan, as a result of the estimation errors imposed by its time cost model. Actually, we also observed this phenomenon in a number of individual queries in the TPC-H experiments: the power-aware optimizer found plans that are superior in both power consumption and actual processing time. It is only in the TPC-C 5GB experiment did we see improvement in the aggregated processing time. This result is positive because the power-aware query optimizer shows even greater potential in power saving than we expected. Its direct outcome is substantial energy savings of 19% and 15.3%.

With the database size increases, more power is consumed for all three systems. This is understandable because more data will have to be read and processed to answer the same query. Buffer hit rate will also decrease when the database is larger. In fact, with the 10GB database, system contention becomes very high – both CPU and disk utilization are close to 100% for most of the time and power consumption goes beyond 40w. The increase of database size has a greater impact on power consumption of the original DBMS: power usage changes from 23.8w to 41.5w in TPC-H and from 37.5w to 42.5w in TPC-C. For the two power-aware DBMSs, the
Table 5.1. Performance and power/energy consumption of the experiments using TPC-H workload and TPC-C workload.

<table>
<thead>
<tr>
<th>DB Size</th>
<th>n</th>
<th>Power (W)</th>
<th>Time (min)</th>
<th>Energy (kJ)</th>
<th>Power Saving</th>
<th>Energy Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-H workloads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5GB</td>
<td>∞</td>
<td>23.8</td>
<td>21.05</td>
<td>30.06</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>21.1</td>
<td>22.55</td>
<td>28.55</td>
<td>11%</td>
<td>5.1%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>20.4</td>
<td>23.88</td>
<td>29.23</td>
<td>14%</td>
<td>2.8%</td>
</tr>
<tr>
<td>1GB</td>
<td>∞</td>
<td>36.6</td>
<td>46.55</td>
<td>106.7</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>31.9</td>
<td>52.05</td>
<td>99.6</td>
<td>13%</td>
<td>6.7%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>30.9</td>
<td>55.29</td>
<td>102.4</td>
<td>16%</td>
<td>4.1%</td>
</tr>
<tr>
<td>10GB</td>
<td>∞</td>
<td>41.5</td>
<td>211</td>
<td>525.4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>34.9</td>
<td>242</td>
<td>508.1</td>
<td>16%</td>
<td>3.3%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>32.4</td>
<td>263</td>
<td>511.3</td>
<td>22%</td>
<td>3.3%</td>
</tr>
<tr>
<td>TPC-C workloads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1GB</td>
<td>∞</td>
<td>37.5</td>
<td>45.34</td>
<td>102.1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>32.9</td>
<td>48.53</td>
<td>95.79</td>
<td>16%</td>
<td>6.1%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>31.1</td>
<td>52.43</td>
<td>97.84</td>
<td>17%</td>
<td>4.2%</td>
</tr>
<tr>
<td>5GB</td>
<td>∞</td>
<td>40.8</td>
<td>123</td>
<td>301.4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>34.2</td>
<td>120</td>
<td>245.5</td>
<td>16%</td>
<td>19%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>32.1</td>
<td>132</td>
<td>254.3</td>
<td>20%</td>
<td>15.3%</td>
</tr>
<tr>
<td>10GB</td>
<td>∞</td>
<td>42.5</td>
<td>223</td>
<td>568.4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>35.9</td>
<td>243</td>
<td>523.8</td>
<td>16%</td>
<td>8%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>33.8</td>
<td>263</td>
<td>533.3</td>
<td>21%</td>
<td>7.7%</td>
</tr>
</tbody>
</table>
average power does not increase as dramatically as that in the original PostgreSQL. As a result, their power savings increase with the size of database.

5.2 Query Characterization (Task 2)

The purpose of this set of experiments is to investigate the power profiles of individual queries. This helps us further understand the sources of power savings in the standard TPC-H workloads. Based on such knowledge, we can design composite workloads with different query components to study the relationship between power/energy consumption and workload characteristics.

We feed the database engine with workloads containing 50 copies of one single query extracted from the TPC-H workload. A problem for running single-query workloads is that it may give unrealistic query processing time due to excessively high cache hit rate. To solve this problem, we duplicate the database into five copies and each query is run on a different copy of the database. By this, we can avoid the effects of data caching in our experiments. There are 22 queries in the TPC-H tools with various features. Among these queries, we picked 19 because the other three are either single-table queries with one single dominant plan or those with extremely long running time in our current test environment. The database size is 1GB.

We compare the power and performance data from running the TPC-H workload with the corresponding estimations given by the (modified) query optimizer, for the purpose of testing the accuracy of such estimations.

Let us first study the estimated power and performance of the plans selected by the query optimizer. From Fig. 5.3a, we can tell that 11 out of the 19 queries show the potential of power saving. They are: queries 1, 3, 5, 6, 7, 8, 11, 13, 15, 17, and 18. For these queries, naturally, the expected power saving comes with the cost of elongated processing time (Fig. 5.3b). However, even with the increase of time, most queries still bear a small energy omitting (Fig. 5.3c). As to those queries that are not expected to provide power saving opportunities, most of them are simple queries (e.g., with joins of only 2-3 tables) that do not have a large search space (of query plans) for the query optimizer to choose from. Thus, the query optimizer ended up choosing the same plan or plans with very similar costs, no matter which \( n \) value we use. For
the ease of future discussions, we classify all the queries into: 1) category I queries: those that are not expected to provide great power saving potential; and 2) category II: those that are likely to show large power savings. An interesting pattern we found is: category I queries are all I/O-bound ones that require a lot of table scanning while most queries in category II are CPU-intensive jobs. This means that, by offering choices of plans with different CPU usage patterns, we can achieve power savings. This, again, supports our claim that CPU plays an important role in the paradigm of power-aware computing.

In comparison, the results of the experiments are interesting. Fig. 5.4a shows that, for all the queries, running the modified DBMS engine leads to power savings to some extent. First, for the category I queries, (which should not show any power saving) there are two cases: 1) those whose execution plans are not changed at all. Their power savings are the results of random system errors at run-time; 2) those whose execution plans are different under different $n$ values. The reason why the running power is lower in the situations of $n = 0$ and $n = 1$ could be (run-time) system modeling errors plus modeling errors in the estimation of energy and performance costs. On the other hand, power savings, although not as significant as those indicated by the estimated values (Fig. 5.3a), can be observed. The real problem is the performance data shown in Fig. 5.4b: when choosing power-only optimizer (e.g., $n = 0$), all queries have unexpectedly long running time, which gives negative energy savings in most cases (Fig. 5.4c). This shows that the power-only query optimizer could be an undesirable choice in the PDBMS design. By comparing the results with the estimated costs in Fig. 5.3, we also see that the method of using multiple copies of the same query has serious limitations in capturing the real (power and time) costs of a query.

While the design of experiments to test the costs of single query is an interesting challenge for future work, we can safely conclude from Fig. 5.4 that power savings do exist. Salient examples are queries 5, 15, and 17, which show savings up to 25%, while a few others showing savings up to 20%. Another lesson learned is: choosing extreme values of $n$ could yield performance that is too poor to bring any direct energy benefit, it is suggested to choose moderate $n$ values in order to get balanced performance and energy.
5.3 Multiple Workloads (Task 3)

In this set of experiments, we again set up 100 virtual clients. First, we use one of them to send queries from the category I pool and the other 99 clients send queries from the category II query pool. Each query in the pool will be picked by equal chance and sent out to the server concurrently. We repeat the experiment under different ratios of clients (ranging from 1:99 to 95:5) from the two query pools. We use the TPC-H benchmark and set the database size to 1GB.

Fig. 5.5 shows the power and performance data normalized by the those of the original DBMS. In Fig. 5.5a, we observe power savings in all 20 cases. However, the savings stay in a relatively steady state as the percentage of category I query increases. On the other hand, relative query processing time has little difference in all workloads (Fig. 5.5b). Thus, we can still see savings on total energy consumption (Fig. 5.5c) in most of the cases. As compared to the power-only \((n = 0)\) optimizer, the one with the more balanced \((n = 1)\) cost model achieved similar power savings yet much better performance, resulting in more energy savings.

With the percentage of category I queries increases, power usage decreases in roughly a linear manner in all three systems (Fig. 5.6a). However, the processing time increases exponentially (Fig. 5.6b) – it increases from about 45 minutes in the 1% case to about 22 days in the 95% case! Naturally, this trend is carried over to the energy consumption data (Fig. 5.6c), although the power decreases.

From these results, one can clearly see that query composition in the workloads does not affect the power saving potential of our query optimizer design. We achieve a considerable margin of power saving despite the dramatic changes of query processing time. By visiting Fig. 5.5a, it seems 20% is a reasonable upper bound for power savings one can reach in any TPC-H workloads.

5.4 Discussions

The above results clearly show significant power savings (up to 22%) of an initial design of PDBMS. The direct energy savings seem to be marginal in most cases. Yet it also reach 15.5%
and 19% in one set of experiments, giving us much enthusiasm to our approach. In summary, we believe our findings are meaningful for the following reasons.

First, even with the marginal energy savings, the economical impact of the envisioned PDBMS will be great. For example, running a single high-performance 300 W server for one year could consume 2628 KWh of energy, with an additional 748 KWh in cooling this server [8]. The total energy cost for this single server would be 338 US dollars a year (for 0.13/KWh) without counting the costs of air conditioning and power delivery subsystems [8]. With our solution, we can save 17-35 US dollars per server every year (considering 50% of total power needed for idling and the 11-22% power savings we can achieve). Again, this does not count the cost savings achieved by lowered hardware failure and cooling demands.

Second, we only used a primitive power-aware DBMS to capture power-saving opportunities. Given the models we have and the tradeoffs we identify, we have reasons to believe that system power usage could be further lowered with more sophisticated cost modeling mechanisms and control algorithms. For example, an adaptive control solution can be designed to guarantee performance by changing the parameters and even the structure of the query evaluation model. Another paradigm we could explore in the foreseeable future is PDBMS running on servers equipped with power-aware hardware, whose power modes can be controlled by the DBMS by designing tunnels that delivers message from DBMS to lower hardware via the OS interfaces. Due to the reasons described in Section 1, our method can be combined with OS-level methods for substantially higher level of power savings.
Figure 5.3. Power and processing time of the chosen plans for the 19 queries in the TPC-H benchmark as estimated by the query optimizer.
Figure 5.4. Power, processing time, and energy consumption of the 19 queries in the TPC-H benchmark as measured by power meter.
Figure 5.5. Relative quantity of power.
Figure 5.6. Absolute quantity of power.
CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we identified power saving opportunities in current DBMS. We argue that query optimization mechanisms in traditional DBMSs could easily miss query plans that are highly power-efficient and yet lead to little degradation of performance. Therefore, to find such tradeoffs was the main task of this study. For that purpose, we performed experiments on the power consumption patterns of various workloads generated from the TPC-H and TPC-C benchmarks in the PostgreSQL system. We believe we have shown a clear picture of the existence of power-performance tradeoffs, and thus the great potential of power conservation in databases. We have observed active power savings up to 22% and total energy savings up to 19% with less than 5% loss in time performance. In the power-aware research field, these are exciting numbers. Another conclusion we can draw is that designing a power-aware query optimizer is a promising direction to enable power conservation: all the above findings are revealed by using a modified PostgreSQL query optimizer equipped with a power-aware query evaluation model. On the other hand, we also demonstrated stable power saving potentials in a wide range of workloads.

Since this thesis serves as a testimony of the economical and technical value of the topic of power-aware data management, abundant future research opportunities can be foreseen. Our vision is to build an unified power-control framework in parallel to the regular query processing modules in the DBMS. Modeling errors in both energy and performance estimation would be a major problem to attack. Specific tasks include refined energy models that capture the dynamics in the system (instead of the static model we used in this thesis); search algorithms in the query plan space; advanced cost evaluation criteria; and more importantly, resource management algorithms to lower the baseline power consumption in the server.
REFERENCES


