Energy Modeling and Management of Database System

by

Yi Zhou

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 5, 2018

Keywords: Energy-efficiency, Modeling, Management, Database system

Approved by

Xiao Qin, Professor of Computer Science and Software Engineering
Wei-Shinn (Jeff) Ku, Associate Professor of Computer Science and Software Engineering
Saad Biaz, Professor of Computer Science and Software Engineering
Richard Chapman, Associate Professor of Computer Science and Software Engineering

Abstract

In this dissertation, we propose a toolkit called EDOM facilitating the evaluation and optimization of energy-efficient multicore-based database systems. Two core components in EDOM are a benchmarking toolkit and a multicore manager to improve energy efficiency of database systems running on multicore servers. We start designing EDOM by analyzing the energy efficiency of two popular database operations (i.e., cross join and outer join) processed on multicore processors. We investigate cross and outer joins, because these two operations are common components of database applications. We describe the criteria and challenges of building an energy efficiency benchmark for databases on multicore servers. We build a benchmarking toolkit, which is comprised of three parts, namely, a configuration module, a test driver, and a power monitor. The workload generator facilitates the configurations of the PostgreSQL database system. We leverage this generator to set up tables and populate data records into the database. The test driver automatically issues operations to the database system in accordance to access patterns created by the workload generator. The power monitor keeps track of energy efficiency and performance of the multicore database system processing the operations driven by the test driver. We develop a multicore manager to optimize the number of cores, thereby making the best tradeoff between performance and energy efficiency in multicore database servers. At the heart of the multicore manager is a memory usage model that estimates memory utilization from queries and database characteristics (e.g., table and record size). An appropriate number of cores is determined using the estimated memory usage to avert unnecessary memory swapping, which induces high energy consumption overhead. We make use of the proposed benchmark toolkit to quantitatively evaluate the performance of our novel multicore manager. Our benchmarking tool of EDOM shows that the multicore and CPU utilizationons have significant impacts on energy efficiency; the cross and outer join operations have remarkable difference in energy consumption; and the indexing technique improves energy efficiency of the database system. More importantly, extensive experimental results show

that our multicore manager in EDOM provides a simple yet powerful solution for improving energy efficiency of database applications running on multicore servers.

In the second part of the dissertation study, we develop an energy-efficient database system called GreenDB running on clusters. GreenDB applies a workload-skewness strategy by managing hot nodes coupled with a set of cold nodes in a database cluster. GreenDB fetches popular data tables to hot nodes, aiming to keep cold nodes in the low-power mode in increased time periods. GreenDB is conducive to reducing the number of power-state transitions, thereby lowering energy-saving overhead. A prefetching model and an energy saving model are seamlessly integrated into GreenDB to facilitate the power management in database clusters. We quantitatively evaluate GreenDB's energy efficiency in terms of managing, fetching, and storing data. We compare GreenDB's prefetching strategy with the one implemented in Postgresql. Experimental results indicate that GreenDB conserves the energy consumption of the existing solution by up to 98.4%. The findings show that the energy efficiency of GreenDB can be optimized by tuning the system parameters, including table size, hit rates, number of nodes, number of disks, and inter-arrival delays.

Acknowledgments

This dissertation would not have been completed without invaluable guidance, experience sharing, constant support and encouragement from my advisor, people in our research group and family members during my study at Auburn University.

First and foremost, I would like to give my most sincere and deepest gratitude to my advisor, Dr. Xiao Qin, for his great efforts, trust and patience in my work. I will never forget his extensive knowledge in the field of computer systems and inexhaustible enthusiasm for research, which keeps inspiring and driving me to accomplish my research. When working on the book chapter "Energy Efficiency of Multicore Database servers" and "GreenDB", his insightful advice and suggestion helped and enlightened me in setting up accurate motivations behind the research, building a EDOM and GreenDB clusters that can reduce energy consumption in database systems.

I am also tremendously grateful to be advised by my committee members Dr.Wei-Shinn (Jeff) Ku, Dr. Saad Biaz and Dr.Richard Chapman who reviewed my proposal and dissertation documents. They gave me a number of valuable suggestions, by which my dissertation had been substantially improved. I would like to show my appreciation for Dr. Zhaofei Fan as my university reader.

Working with our research group is fantastic. I owe my gratitude to Shubbhi Taneja, Chaowei Zhang, Yuanqi Chen, Xiaopu Peng, Yangyang Liu, Jianzhou Mao, Ajit Chavan, who helped me with paper writing, experimental result collection and group discussions. In addition, all the professors and students in the Department of Computer Science and Software Engineering are greatly appreciated, because an excellent atmosphere for study and research is created and maintained by everyone.

Finally and most importantly, the endless love from my family is the most powerful strength that keeps me fighting for my research. My mother Xiancui Liu, my father Ziyou Zhou always stay with me, cheering for achievement and overcoming all difficulties.

To my parents, I really appreciate all your support and encouragement while I am working on my Ph.D these three years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

High energy efficiency is of importance for reducing operating cost of data centers, where database applications are running on multicore servers [35]. Traditional energy saving techniques for database systems are inadequate for multicore computing. To address this problem, we propose in this study a multicore manager called *EDOM* - a simple yet effective way of improving energy efficiency of database operations on multicore servers. To investigate energy efficiency of multicore database systems, we build an energy-efficiency benchmarking toolkit for modern database systems. We show that the toolkit can be applied to evaluate the energy efficiency of our proposed EDOM on multicore servers.

Improving energy efficiency of parallel database systems also becomes indispensable for building data centers supporting transnational database applications. Traditional energy-saving techniques for clusters are inadequate for parallel database systems running on clusters. To address this problem, we focus on reducing energy consumption cost of large-scale database systems, which are comprised of multiple nodes or severs. We show how to develop a parallel database system called *GreenDB* - an energy-effective system running on clusters. At the heart of *GreenDB* is a prefetching and caching mechanism, which fetches hot data from passive nodes into active nodes. *GreenDB* offers energy savings by turning a large number of passive nodes into the low-power mode, which keeping a small number of active nodes to serve queries.

The power management of *GreenDB* is governed by two mathematical models, namely, a prefetching model and an energy consumption model. We show how to build these two models and how to integrate the models into the prefetching and caching mechanism.

1.1 Motivations

The following motivations make energy-efficiency benchmarking tools and energy-efficient prefetching and caching in GreenDB desirable and achievable.

1. The lack of study on the energy efficiency of database operations (e.g., cross and outer joins) running on mulitcore servers.
2. The pressing need of benchmarking tools for energy-efficient database systems.
3. The growing importance of improving energy efficiency of database systems through multicore management.
4. Database clusters are cost-effective platforms for parallel database systems.
5. A pressing demand of reducing energy cost of large database systems.

Computer architecture enters a new era of multicore structures , which become a standard computing platform for a wide range of application domains including database systems [59]. A key concept of multicore computing lies in multi-threading, which concurrently executes multiple threads on multiple cores. Although growing attention has been paid to the improvement of database energy efficiency [31], little attention has been paid to the energy efficiency analysis of database operations on multicore processors widely deployed in modern servers. Hence, we are motivated to kick off this research by focusing on the analysis of energy efficiency of database operations running on multicore servers.

To optimize energy efficiency of database systems, one has to rely on benchmarks to assess the effectiveness of energy-saving schemes deployed in the systems. In a handful of prior studies, benchmarks have been developed for energy efficiency in data centers. For example, JouleSortis used to evaluate the energy efficiency of clusters. The existing benchmarks were focused on energy efficiency of cluster computing systems rather than database systems. This problem inspires us to develop an energy-efficient benchmarking tool for database operations processed in multicore systems.

The hardware advancement of multicore processors brings new challenges to the design of database systems, because a main performance bottleneck shifts from slow I/O access to main memory access [12]. This challenge becomes more pronounced for data-intensive applications like database processing. This challenge motivates us to investigate how to choose an appropriate number of cores to improve energy efficiency of multicore database systems by averting the memory bottleneck problem.

Database clusters have become a cost-effective computing platform to manage a massive amount of data for database management systems or DBMS [61][49][44]. Clusters achieve high-performance and low response time for database applications through parallel query processing on multiple database servers [34]. Data in clusters are partitioned and replicated among database nodes to handle queries in parallel.

Database Clusters - cost-efficient platforms - consist of multiple DBMS nodes being independent of one another. It is not uncommon for database clusters to employ a middleware layer to coordinate parallel queries evenly distributed among database nodes. Such a middleware layer can easily and efficiently fetch and migrate data managed in the database nodes, because indexing large tables faciliates data migrations when data are replicated across multiple database nodes [50].

Energy cost is one of the significant components of operational costs in data center environments [37]. Evidence shows that a data center containing 1000 racks consumes 10MW total power per year [43]. A wide variety of techniques were proposed to build high-performance and energy-efficient clusters in data centers, because it is greatly desirable to facilitate energy-efficient and environmental friendly clusters [67]. Unfortunately, we observe that little attention has been paid to energy efficiency improvement of database systems running on clusters. This observation motivates us to focus on energy-efficient database systems for modern data centers.

A handful energy-saving techniques were designed for single-server database systems. For example, Chehadeh *et al.* investigated energy-efficient indexing strategies in an object-oriented database environment [8]. Poess and Nambiar proposed a way of making tradeoffs between existing power-saving techniques and their performance impact on database applications. Our analysis will guide system developers and data center managers in making informed decisions

regarding adopting power-preserving techniques. The aforementioned energy-saving schemes developed for single-server database systems are inadequate for DBMS on clusters. Moreover, energy-efficient prefetching and caching have not been incorporated in the prior database systems. We address this challenge by proposing an energy-efficient database cluster called *GreenDB*, in which prefetching and caching are seamlessly integrated to offer energy savings in a holistic way.

1.2 Contributions

We make the following six contributions in this study.

1. A toolkit called *EDOM* facilitating the evaluation and optimization of energy-efficient multicore-based database systems.
2. An analysis of energy-efficiency impacts of multicore processors on database operations.
3. An energy-aware multicore manager - a core component in *EDOM*.
4. A energy efficiency architecture called *GreenDB* prefetch and cache the most frequently used data table in the hot nodes.
5. A prefetching model - a core component in *GreenDB*.
6. Energy saving prediction model is deployed in a distributed PostgreSQL Database System to evaluate the energy efficiency.

1.3 Organization

The rest of this dissertation is organized as follows. The next chapter presents prior studies and related research projects. In Chapter 3, we propose a toolkit called *EDOM* facilitating the evaluation and optimization of energy-efficient multicore-based database systems. Criteria and challenges of building an energy efficiency benchmark is described. A multicore manager aiming to determine the optimal number of cores is designed and a memory usage model relying

on queries and database characteristics is proposed. This multicore manager is validated by our *EDOM* benchmark toolkit.

In Chapter 4, an energy-efficient database system called *GreenDB* running on clusters is designed and developed by the virtue of applying a workload-skewness strategy to manage hot nodes coupled with a set of cold nodes in a database cluster. Chapter 4 also introduces an energy-cost model estimating energy savings offered by keeping candidate data tables into the hot nodes while shutting down the cold nodes. In this Chapter (see Chapter 4.2), we shed light on the implementation of an asynchronous prefetching scheme in GreenDB, which takes a full advantage of asynchronous replications to cut energy cost of fetching hot data from cold nodes. Moreover, we evaluate the effectiveness of our GreenDB clusters by conducting a sets of experiments under various situations (see Chapter 4.3).

Finally, Chapter 5 concludes this dissertation research by summarizing key contributions (see Chapter 5.1). Future research directions can be found in Chapter 5.2.

Chapter 2

Related Work

Energy-efficient clusters are becoming increasingly popular in large-scale data centers to reduce high operational cost caused by huge energy consumption. Recent studies proposed a wide range of energy-saving techniques in the realm of cluster computing.

Evidence shows that a variety of factors affect the energy-efficiency of DMBS. For example, Tsirogiannis *et al.* analyzed the energy efficiency of processors in a database server, discovering that energy consumed by CPUs does not vary linearly with CPU utilization under database workload [57]. Lang *et al.* analyzed a number of important parameters related to the design of energy-efficient DBMS [31]. A framework built by Lang *et al.* optimizes queries by considering both performance and energy consumption as optimization criteria [31].

Increasing attention has been paid to making good tradeoffs between energy efficiency and performance in the field of DBMS. Although energy saving and high performance are two conflicting design goals, prior findings indicate that a few energy-efficient configurations and schemes may deliver good performance and achieve high energy efficiency [57]. Schall and Härder developed a distributed DBMS - WattDB - to make dynamical configurations to satisfy performance demands while conserving energy consumption on clusters [52]. Xu *et al.* implemented PET - an energy-aware query optimization framework that is an extension of the PostgreSQL kernel [62]. PET makes use of its power cost estimation module and plan evaluation model to allow database system to make good tradeoffs between energy efficiency and performance. Our EDOM is distinctly different from the aforementioned approach in that EDOM aims to conserve energy cost of DBMS running on multicore servers while achieving high query performance.

Recently, some research begun to pay attention to model the energy consumption of the database systems. Rivoire described different approaches to modeling power consumption in components, systems, and data centers, aiming at improving the components' and systems' design or to energy efficiently use existing hardware. A few models were focused on profiling disk energy behavior based on CPU demands [64]. Different from the existing energy consumption models, our models(i.e., data fetching model and power saving model) built in GreenDB pay attention to modeling energy saving based on data transfer between hot node and cold nodes.

2.1 Energy-Efficient Data Centers

Tens of thousands of data centers around the world are consuming huge amount of energy. Increasing business companies, IT companies, and institutes are planning to build their own data centers. A study by DatacenterDynamics demonstrates that worldwide investment in data centers in 2012 had increased by 22.1% up to 105 billion dollars compared with 2011, and this investment is going to grow by another 14.5% to 120 billion dollars for 2013 [27].

Research shows the rapid increment of energy consumption of data centers [21] [26] [55]. A report announces that 1,500 TWh of electricity, which is nearly 10% of world electricity generation, is used by the world's Information-Communication-Technologies (ICT) ecosystem annually [39]. Furthermore, global data centers are estimated to consume (as of 2010) from 250 to 350 TWh every year. A reason behind the striking energy consumption in data centers is the rapid growth of computing and storage capacity in recent years. For instance, Facebook has invested more than 1 billion in IT facilities that power its social network, which now serve more than 845 million users in a month around the world [38].

Cloud computing has become a popular topic in recent years. A study shows that coal and nuclear, which generate severe air pollution, are used to satisfy these large amount of electrical energy demand [14]. Apple, HP, IBM, Facebook, and Mircosoft are using dirty energy to power their growing cloud data centers. Confronting with the rapid increment of energy consumption and severe air pollution, growing attention has been paid to build energy-efficient data centers [2] [16] [22] [32]. At the same time, small or medium sized organizations began to

move their computing applications to an Internet-based ”cloud” platform in order to improve energy efficiency [60].

Computing cost and cooling cost are major components of total energy consumption for data centers. Computing cost refers to the electronic energy cost that makes the IT facilities working. And cooling cost is the cost of cooling systems that lower down the temperature in data centers. Studies have been conducted on reducing either the computing cost or cooling cost in order to build energy-efficient data centers.

2.1.1 Computing Cost

A lot of research have been done in reducing computing cost of data centers [4] [53] [63]. For instance, CMPs are widely used in data centers, and the frequency/voltage of CPU cores could be adjusted in order to save power consumption. Mishra *et al.* proposed a two-tier feedback-based control scheme, in which the first-tier is comprised of a global power manager to allocate power targets to individual islands according to workloads and the second-tier consists of local controllers that adjust island power through changing the voltage and frequency as a response of workload requirements [40]. A power-efficient scheme for erasure-coded storage clusters–ECS2–was proposed, which aims to offer high energy efficiency with marginal reliability degradation [25].

Popular strategies to reduce computing cost include redistributing workload and powering off idle disks or data nodes. For example, an energy-efficient strategy was proposed which specifies a subset of disks as cache disks and dispatches workloads to these cache disks while making the other disks spin down [13]. Another strategy introduced a Popular Data Concentration (PDC) technique that migrates frequently accessed data to a subset of disks [45]. Then the other disks which are not accessed frequently could be transitioned to low-power mode, and the total computing cost of these data nodes could be reduced.

Many researchers concentrate on resource management and task scheduling in data centers to decrease computing energy consumption [3] [5] [7] [33] [56]. For instance, Beloglazov and Buyya proposed an energy-efficient resource management system for virtualized Cloud data centers [7]. In this system, VMs are consolidated according to the utilization of resources,

and virtual network topologies are built between VMs and thermal status of computing nodes to save energy. This management system reduces the operational costs of data centers and provides the required Quality of Service (QoS). Beloglazov *et al.* also demonstrated an architectural framework (including resource provisioning and allocation algorithms) and principles for energy-efficient Cloud computing [5]. Experimental results show that their Cloud computing model has immense potential in energy saving and energy efficiency improvement under dynamic workload scenarios. In addition, Aksanli *et al.* demonstrated an adaptive job scheduler that utilizes the prediction of solar and wind energy production [3]. This job scheduler improves the energy efficiency by three times. Lee and Zomaya pointed out that under-utilized resources account for a large amount of energy use and resource allocation strategies could be applied to achieve high energy efficiency [33]. They proposed two task consolidation heuristics methods that aim to maximize resource utilization and take into account of both active and idle energy consumption. Experimental results illustrated the energy saving capability of their heuristics.

With the growing of data center density and size, designers should take into account of both energy costs and carbon footprint. Altering the usage patterns of data centers is believed to be a practical method to affect demand response. Chiu *et al.* pointed out that shifting computational workloads across geographic regions to match electricity supply may help balance the electric grid [10]. They proposed a symbiotic relationship between data centers and grid operators and a low cost workload migration mechanism. Ren and He proposed an online algorithm, called COCA (optimizing for COst minimization and CArbon neutrality), for minimizing operational cost in data centers while satisfying carbon neutrality without long-term future information [48]. COCA enables distributed server-level resource management: each server autonomously adjusts its processing speed and optimally decides the amount of workloads to process. Analysis of trace-based simulation studies show that COCA reduces cost by more than 25% (compared to state of the art) while resulting in a smaller carbon footprint.

Furthermore, network facilities are also investigated in order to reduce the energy consumption of data centers. The architecture of a Data Center Network (DCN) affects its scalability, however, its power consumption is a main contributor to its energy cost. Hammadi

and Mhamdi classified existing DCNs as switch-centric and server-centric networks, and conduct literature review of existing technologies in energy saving and renewable energy approaches [23].

2.2 Factors Affecting Energy-Efficiency

Prior studies revealed that energy-efficient database systems are affected by various factors. For example, Graefe's novel technique makes a server-class database management system energy efficient [6]; Graefe's findings advocate that I/O scheduling and placement offer opportunities for energy savings. Wang *et al.* recently investigated energy-saving data management techniques, indicating that data management software plays a vital role in boosting energy efficiency [19]. Theo *et al.* built a cluster framework, where individual nodes are dynamically attached and detached to a cluster on demand of dynamic workload [51].

Attention has been paid towards balancing energy efficiency and performance in the realm of database systems. Although energy savings and high performance are two conflicting design goals, our previous research [66] suggests that in a single database server, energy-efficient configurations may deliver good performance and high energy efficiency thanks to a significant reduction in idle time.

2.3 Energy-Saving Techniques

Improving energy efficiency becomes increasingly important for data centers. Techniques or strategies reducing energy cost make a major contribution to advance energy-efficient data centers.

Xu *et al.* implemented an energy-aware query optimization framework or PET as an extension of the PostgreSQL kernel [62]. PET makes use of its power cost estimation module and plan evaluation model to make good tradeoffs between energy efficiency and performance in database systems. Our *GreenDB* is distinctly different from the aforementioned approaches in that GreenDB conserves energy cost of distributed DBMS running on multiple servers while achieving high query performance through the data management architecture.

Very recently, researchers embarked on inspiring studies to model the energy consumption of database systems. Rivoire proposed approaches to modeling power consumption in components, systems, and data centers, aiming to improve energy-efficient system design [54]. A handful models were focused on profiling disk energy behaviors that depend on CPU demands [64]. Different from the existing energy consumption models, ours (i.e., the prefetching and power-saving models) built in GreenDB pay attention to modeling energy savings contributed by transferring popular tables from cold to hot nodes.

2.4 Energy Consumption Model

Recently, energy consumption models have been developed for computing systems [9]. A few models were focused on projecting energy consumption of disk systems using CPU demands. The energy consumption models for storage systems are inadequate for DBMS. Other intriguing high-level energy models were proposed for DBMS [24]. For instance, Poess and Nambiar constructed a power consumption model using the TPC-C benchmarks; the models were validated with measurements taken from three TPC-C configurations that are comprised of client systems, a database server, and a storage subsystem [46]. Different from the existing energy consumption models, our models built in EDOM pay attention to modeling energy consumption of DBMS operations.

2.5 Energy-Efficient Database Management Systems

Prior studies revealed that energy-efficient database systems are affected by various factors. For example, Graefe's novel technique makes a server-class database management system energy efficient [6]; Graefe's findings advocate that I/O scheduling and placement offer opportunities for energy savings. Wang *et al.* recently investigated energy-saving data management techniques, indicating that data management software plays a vital role in boosting energy efficiency [19]. Theo *et al.* built a cluster framework, where individual nodes are dynamically attached and detached to a cluster on demand of dynamic workload [51].

2.6 Summary

One objective of this dissertation is to propose energy-aware management strategies to save energy cost of data centers. To reduce energy consumption, efforts were placed on improving energy efficiency in data centers. In the first section of this chapter (see also Chapter 2.1), we introduced popular methods in building energy-efficient database systems. In the second section, we discussed energy-aware data management of database clusters. We observed that CPUs are an energy consumption dominant component contributing most portion of the overall energy consumption of database systems. In addition, evidence shows that memory plays a critical role in energy efficiency of the database systems.

Related studies confirm that database clusters have been wildly adopted in a wide range of applications. We presented in Chapter 2.5 the related work on modeling energy savings of caching data tables in active nodes. Then, we illustrated prior studies on asynchronous energy-efficient prefetching schemes. Finally our briefly introduced energy-aware management strategies.

Chapter 3

Energy Efficiency of Multicore Database servers

In this chapter, we propose a toolkit called *EDOM* facilitating the evaluation and optimization of energy-efficient multicore-based database systems. Two core components in *EDOM* are a benchmarking toolkit and a multicore manager to improve energy efficiency of database systems running on multicore servers.

We start this study by analyzing the energy efficiency of two popular database operations (i.e., cross join and outer join) processed on multicore processors. We investigate cross and outer joins, because these two operations are common components of database applications.

We describe the criteria and challenges of building an energy efficiency benchmark for databases on multicore servers.

We build a benchmarking toolkit, which is comprised of three parts, namely, a configuration module, a test driver, and a power monitor. The workload generator facilitates the configurations of the PostgreSQL database system. We leverage this generator to set up tables and populate data records into the database. The test driver automatically issues operations to the database system in accordance to access patterns created by the workload generator. The power monitor keeps track of energy efficiency and performance of the multicore database system processing the operations driven by the test driver.

We develop a multicore manager to optimize the number of cores, thereby making the best tradeoff between performance and energy efficiency in multicore database servers. At the heart of the multicore manager is a memory usage model that estimates memory utilization from queries and database characteristics (e.g., table and record size). An appropriate number of

cores is determined using the estimated memory usage to avert unnecessary memory swapping, which induces high energy consumption overhead.

We make use of the proposed benchmark toolkit to quantitatively evaluate the performance of our novel multicore manager. Our benchmarking tool of *EDOM* shows that the multicore and CPU utilizationons have significant impacts on energy efficiency; the cross and outer join operations have remarkable difference in energy consumption; and the indexing technique improves energy efficiency of the database system. More importantly, extensive experimental results show that our multicore manager in *EDOM* provides a simple yet powerful solution for improving energy efficiency of database applications running on multicore servers.

The rest of the chapter is organized as follows. In Section 3.1, the criteria and challenges in the development of our energy efficiency benchmark are discussed. The design of energy-efficiency benchmark as well as implementation issues are stated in Section 3.2. Section 3.4 provides an investigation in operations regarding energy and performance efficiency, followed by a detailed analysis. Finally, Section 3.5 concludes this part of the dissertation study.

3.1 Criteria and Challenges

Noticing that there is the lack of simple yet efficient benchmarks for energy-aware database systems, we start this study by focusing on the criteria and challenges of the development of energy efficiency benchmarks in the realm of database. The criteria presented in this section set the preliminary principles by which our energy efficiency benchmark is established.

Existing energy management studies (e.g., [31]) paid attention to the energy-efficiency evaluation and comparison of energy-efficient database management systems. In contrast, the first part of our study is focused on energy-efficiency benchmarking tools that address issues related to energy profiling, energy efficiency, and continuously changing performance.

3.1.1 Energy Efficiency Profiling.

Ideally, an energy-efficiency benchmark should offer us an intuitively profiling approach by which we can directly test, measure, and analyze a database system's energy efficiency [42].

Our tool aims to show that multicore processors, CPU utilization, memory usage, and hard disks affect energy consumption when the database system executes queries.

Energy-efficiency profiling benchmarks provide two remarkable benefits. First of all, one can take full advantage of energy-efficiency profiling to establish an energy-efficiency model which mathematically demonstrates the correlations among multicore processors, CPU utilization, memory usage, and indexing [36]. Secondly, energy-efficiency profiling provides us with the ability to facilitate an effective estimation of new techniques deployed to improve the energy-efficiency of database systems.

In short, energy-efficiency benchmarks make it possible to investigate energy cost caused by hardware (e.g., multicore processors, CPU utilization, and memory usage) and software components (e.g., indexing, query types, optimization strategies).

3.1.2 Consumption Measurement.

The goal of our benchmarking tool is to measure real-world database systems deployed in modern data centers. There are three salient features of our tool.

• First, to fulfill testing needs, one can configure workload conditions by varying table size and choosing indexing schemes.

• Second, the test-driver tailored for a database system is able to choose the query type, amount of execution time, and CPU utilization rate.

• Third, our tool offers a simple yet efficient way of testing energy-efficient database systems. It is straightforward to apply the tool to automatically and concurrently measure and record both the power consumption and performance.

3.1.3 Performance Profiling.

Measuring database system performance such as execution time, response time, and throughput should be taken into account in energy-efficiency benchmarks. On one hand, reducing the energy cost in modern data centers is important and indispensable [29]; on the other hand, improving system performance is a crucial aspect of metrics to evaluate overall database efficiency [11]. In a vast majority of cases, system administrators have to make tradeoffs between

energy efficiency and performance. For certain applications, it is not worthwhile to conserve energy at the cost of a significant performance degradation [15]. An energy-efficiency benchmark should be able to measure energy efficiency in conjunction with performance of database systems. An ideal energy-efficiency benchmarking tool offers constructive guideline for system administrators to improve the energy efficiency while maintaining database performance in large data centers.

3.1.4 Optimization for Multicore-based Database Systems.

It is challenging to optimize database systems running on multicore processors [65]. This challenge becomes even more daunting when it comes to making energy cost and performance tradeoffs for database systems running on multicore servers. We pay attention to multicore management to optimize energy efficiency of cross join and outer join operations running in multicore systems. We design a multicore manager to optimize the number of cores in order to make good tradeoffs between performance and energy efficiency in multicore database servers.

The challenge of developing the multicore manager lies in a memory usage model [30], which is responsible for estimating memory utilization using queries and database characteristics like table and record size. The multicore manager decides how many cores should be allocated to process database operations without giving rise to the memory swapping problem that causes high energy consumption [17]. The multicore manager and the benchmarking tool should be seamlessly integrated into the EDOM system for energy-efficient database systems. As such, we can apply the benchmark toolkit to quantitatively evaluate the performance of the multicore manager.

3.2 The Energy Efficiency Benchmark

In this section, we present the design issues of the energy efficiency benchmark toolkit for database systems. Our benchmark tool plotted in Fig. 3.1 consists of three components, namely, the configuration module, the test driver, and the power-performance monitor.

3.2.1 Simplicity of EDOM toolkit

The module-oriented design approach gives rise to the simplicity of our EDOM toolkit. To make our EDOM toolkit easy and portable to use, we group the functions into three modules, which consist of related source code. Furthermore, to reduce the overhead of the energy consumption and computing source, the test driver and power-performance monitor (see Fig. 3.1) are implemented by a highly light-weighted scripting language (i.e., Python). In addition, the configuration module is comprised of three submodules, namely, query type generator, CPU utilization configuration, and multi-core utilization configuration. The configuration module is launched only once to set up the test driver before EDOM is kicking off to evaluate and optimize the of energy-efficiency of database systems. After the configuration is completed, the module requests no system resource.

3.2.2 Configuration Module

The responsibility of the workload generator is three-fold. First, it configures the table size of tested database according to an experiment design. Second, the module can enable or disable indexing features during the course of energy efficiency testing. Last, the module provides a straightforward way of managing data of tested tables in a database system. The configuration module automatically creates and setups a large amount of test data imported to the database system prior to a test. The configuration module also adjusts field types and sizes with accordance to specific test requirements.

3.2.3 Test Driver

The test driver generates a set of queries issued to the tested database system. This module contains three parts: the query type generator, the CPU utilization controller, and the multicore controller.

The query type generator manipulates the types of performed queries in the PostgreSQL database. The CPU utilization controller configures the CPU utilization of a server processing all the issued queries. For example, this controller can set the CPU utilization to four different

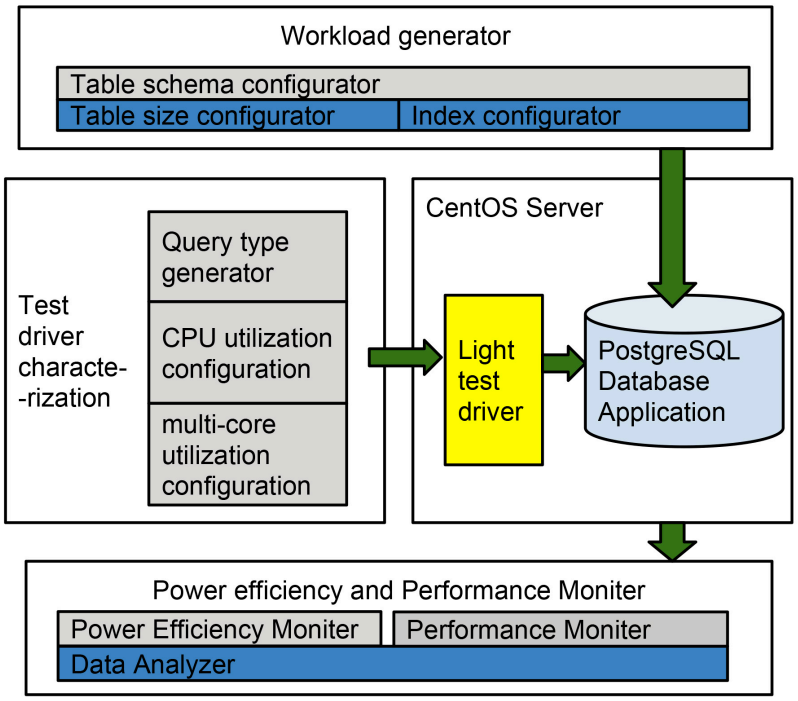


Figure 3.1: The framework of the energy efficiency benchmarking toolkit, which consists of a configuration module, a test driver, and a power-performance monitor.

levels (i.e., 25%, 50%, 75%, and 100%). The multicore controller is in charge of setting the number of cores running queries. For instance, in our experiments, the number of cores can be flexibly configured to a number anywhere between one and four.

3.2.4 CentOS and PostgreSQL

We run the PostgreSQL database system on CentOS. PostgreSQL, an object-relational database management system with high extensibility, securely stores and retrieves data for other software applications [58]. ProstgreSQL is capable of processing workloads of small-scale applications as well as large Web-based applications.

In addition, we maintain a dedicated computing environment to test PostgreSQL, because the focus of our experiments is to measure energy consumption of database operations. Query requests are issued by the light-weight test driver, the energy consumption of which is ignored in our experiments. Table 4.4 shows the database server specification.

Table 3.1 Testbed Configurations.

	OptiPlex 3020 MT/SFF Technical Specifications
CPU	Intel 4th Core i5-4570 Quad Core@3.20GHz
Memory	4GB Non-ECC 1600MHz DDR3 SDRAM
Hard Drives	Seagate KC47-500GB SATA (7.200 RPM)
Operating System	CentOS 6.5 (Final) Linux kernel $2.6.32 - 431.el6.x86_64$
Database System	PostgreSQL 9.3.5

3.2.5 Power Efficiency and Performance Monitor

The power-performance monitoring module is responsible for measuring and collecting metrics like power consumption, processing time, CPU utilization, and memory usage [41]. We apply an electricity meter to measure power consumption of a power outlet socket, to which our server is connected.

To improve the measurement accuracy, we connect the display into another power socket, ensuring that the measured energy is only consumed by the PostgreSQL database server. The power meter employed in this study is TS-836A Plug Energy Watt Voltage Amps Meter (see Table 4.2 for details).

In addition to energy consumption, the other measured metrics such as processing time and memory usage are automatically collected by a light-weight process implemented in a Python script.

Table 3.2 Power Meter Specifications

	TS-836A Power Meter Specifications
Measurement of consumption	$0.00 \sim 9999.99$ KWh
Voltage display range	$0V \sim 9999$V
Current range	$0.000A \sim 15.000$A
Frequency display	$0Hz \sim 9999$Hz
Wattage display (Watts)	$0 \sim 1800$W

3.3 The Multicore Manager

In this section, we propose a multicore manager - a core component of *EDOM*. The goal of the multicore manager is to make a good tradeoff between energy efficiency and performance in

database systems. This goal is achieved by alleviating the memory swapping problem through the decisions on the most appropriate number of cores.

We design a memory usage estimator to provide a guideline for determining the number of cores (see Section 3.3.1). We show the algorithm of the multicore manager in Section 3.3.2. The energy efficiency of a database system governed by the multicore manager is evaluated in Section 3.3.3.

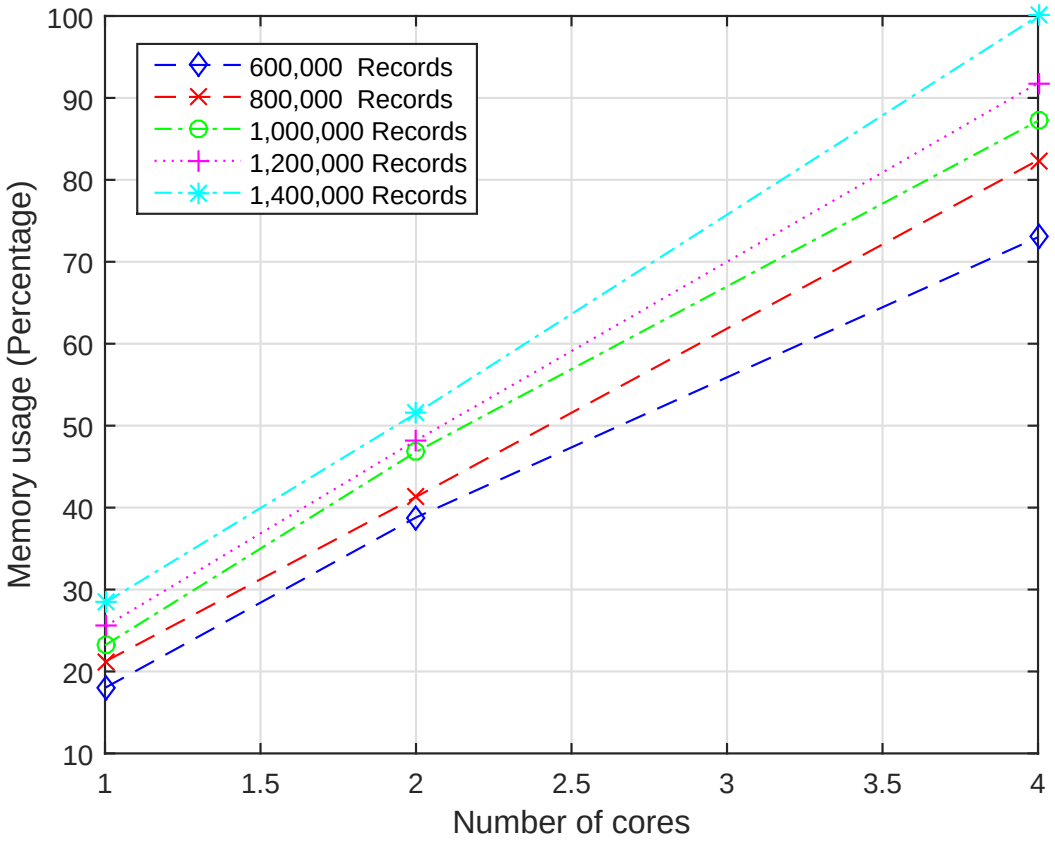


Figure 3.2: The impact of the number of cores on memory usage of a multicore-based database system.

3.3.1 Memory Usage Estimator

The optimal number of cores utilized in a multicore-based database system largely depends on workload conditions (e.g., query types, data size, and processing time). The workload conditions exhibit various memory-usage characteristics, which in turn affect the optimal number

of cores employed in the system. This observation motivates us to develop a memory usage estimator to provide a general guideline for determining the number of cores.

In modern database systems, memory resources become a vital component affecting performance and energy efficiency [18]. This argument is especially true when it comes to big data applications. When a database system has insufficient free memory, then some memory resources must be freed by writing data back to disks [20].

Our empirical study reveals that the memory usage imposes a significant impact on the energy efficiency of database systems. For example, Figs. 3.2, 3.5(a) indicate that heavily utilized main memory adversely slows downs the performance of multiple cores; as a result, an increasing number of queries cannot be processed in a timely manner, thereby pushing the power consumption at an unacceptably high level.

The multicore manager aims to decide an optimal number of cores that meets the resource needs of heavy workload, where main memory becomes scarce resources. Recognizing that the optimization of number of cores relies on memory utilization, we develop a memory usage estimating module to predict memory utilization under any workload modeled in forms of query types and the other database characteristics. And we designed an effective algorithm to calculate the most appropriate number of cores for running operations under a limited memory hardware situation.

To address the problem of heavy workload coupled with scarce memory resources, the memory usage estimator estimates the memory usage according to the following four operation factors.

- database query type,
- the number of tables,
- the number of records, and
- the record size.

Fig. 3.3 depicts the architecture of the multicore manager, which consists of five key components. Given the number of tables, the number of records in the table, and record size, the

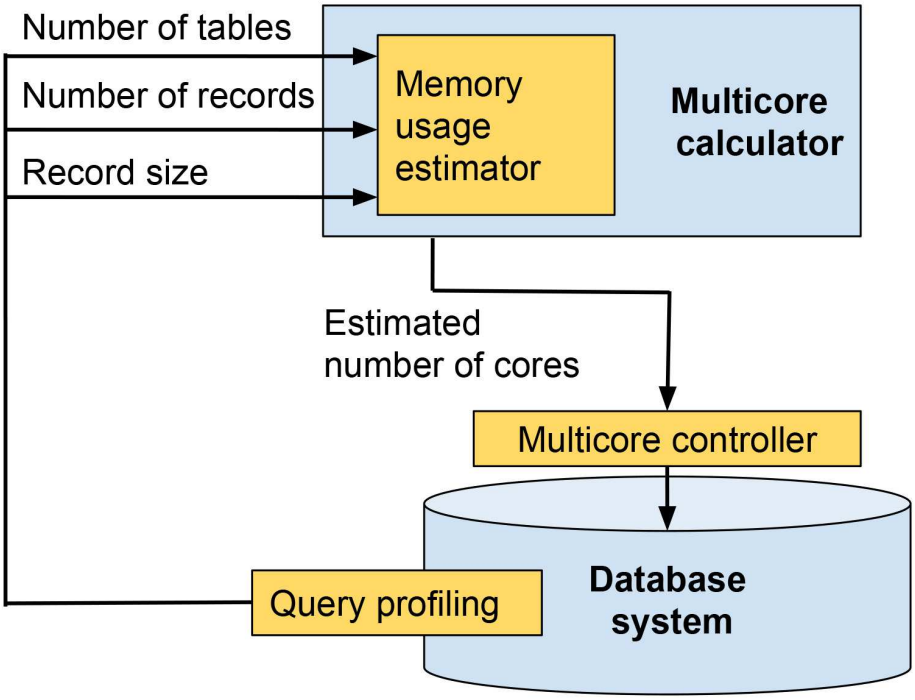


Figure 3.3: The framework of the memory usage estimator.

memory usage estimator applies a mathematical model to project memory usage, which is used in the multicore calculator to determine the number of the mulitcores.

The multicore manager algorithm detailed in Section 3.3.2 incorporates a multicore calculator (see Fig. 3.3) to govern the process of choosing an approximate number of cores. The query profiling in the multicore manager is designed to acquire the database workload characteristics such as the numbers of tables and records. The multicore controller obtaining the estimated number of cores is in charge of setting up the number of cores that execute database queries in the system.

3.3.2 Algorithm Design

In this section, we propose the algorithm of the multicore manager to optimize the number of cores for given workload conditions. The primary function of the multicore manager is to make a good tradeoff between energy efficiency and performance in database systems running on multicore servers.

Recall that the memory usage estimator (see Section 3.3.1) predicts memory utilization from queries and database characteristics. An appropriate number of cores is determined by Algorithm 2, which takes an estimated memory usage as the input to alleviate the memory swapping problem.

Algorithm 1 Multicore Manager Algorithm: $optimal()$

Require:
number of tables t
number of records r
record size s
Ensure:
number of cores C_{opt}
1: $C_{min} \leftarrow 1$;
2: $C_{max} \leftarrow MAX\_NUM\_CORES$;
3: $core\_search(C_{min}, C_{max}, t, r, s)$;
4: **return** C_{opt};

The multicore manager algorithm (see Algorithm 1) outlined above initializes the minimal and maximal number of cores to C_{min} and C_{max}, respectively (see Lines 1-2 in Algorithm 1). Next, the multicore manager algorithm invokes the binary search algorithm (see Algorithm 2) to recursively calculate the number of cores under a workload condition expressed in the form of the number of tables t, the number of records r, and the number of record size s (see Line 3 in Algorithm 1).

Algorithm 2 Recursive Core Search: $core\_search()$

Require:
C_{min} **- Minimal number of cores**
C_{max} **- Maximal number of cores**
number of tables t
number of records r
record size s
Ensure:
number of cores C_{opt}

1: $C_{opt} = \lfloor \frac{Cmin+Cmax}{2} \rfloor$
2: **if** $(memory_{estimated}(t, r, s, C_{opt}) < memory)$ **then**
return $core\_search(C_{opt}, C_{max}, t, r, s)$;
3: **else if** $memory_{estimated}(t, r, s, C_{opt}) > memory$ **then**
return $core\_search(C_{min}, C_{opt}, t, r, s)$;
4: **else**
return C_{opt};
5: **end if**

The binary core search algorithm is recursive in nature. In each recursion, the algorithm takes the following two main steps to obtain the appropriate number of cores.

- **Step 1.** The optimal number of core C_{opt} is set to the midpoint between the minimal (i.e., C_{min}) and maximal (i.e., C_{max}) numbers of cores (see Line 1).

- **Step 2.** Using workload condition (i.e., t, r, s) and the tentative optimal number of cores C_{opt}, the memory usage estimator predicts the memory load (i.e., $memory_{estimated}(t, r, s, C_{opt})$) (see Line 2 in Algorithm 2).

- **Step 3.** If the projected memory load is smaller than the available memory capacity, then the optimal number of cores is increased by recursively calling the $core\_search()$ algorithm, where the searching range is between C_{opt} and C_{max} (see Line 2 in Algorithm 2). Otherwise, when the estimated memory load exceeds the available memory size, $core\_search()$ is recursively invoked to update the optimal core number by searching a range between C_{min} and C_{opt} (see Line 3 in Algorithm 2).

We demonstrate in the next subsection (i.e., Section 3.3.3) that an appropriate number of cores determined using the multicore manager algorithm helps in alleviating the serious memory swapping problem - a main driver for high energy cost in multicore database systems.

3.3.3 Energy-Efficient Multicore Manager

In order to validate the effectiveness of our multicore manager, we design a group of experiments. We also quantitatively measure the energy efficiency of the multicore manager.

In this set of experiments, we increase the table size from 6×10^5 records to 1.4×10^6 records with an increment of 2×10^5 records. We evaluate the power consumption of a database system, where the number of cores is dynamically controlled by our multicore manager (see Algorithm 1). We compare our algorithm with two baseline solutions, where the number of cores is fixed to one core and four cores.

Fig. 3.4 shows the energy consumption of the multicore-based database system governed by our multicore manager; Fig. 3.4 also illustrates the energy consumption in the one-core and four-core cases.

The results indicate that as the table size is increased from 6×10^5 to 1.2×10^6 records, the optimal number of cores in terms of energy efficiency is four; in such a relatively light load, the

one-core manager exhibits the worst energy efficiency. The evidence shows that our multicore manager chooses the optimal number of cores to reduce energy consumption under low and medium-low workload conditions.

When the table size is very large (e.g., 1.4×10^6 records), the energy consumption of the four-core case dramatically increases due to the memory-swapping problem (see also Fig. 3.4). Not surprisingly, our multicore manager judiciously downgrades the number of cores to three, which significantly conserves energy consumption compared with the four-core case. Moreover, the results demonstrate that our multicore manager is also more energy efficient than the one-core counterpart. We observe from this group of experiments that the multicore manager is capable of determining an optimal number of cores under relatively heavy workload (e.g., table size larger than 1.4×10^6 records).

We conclude that the multicore manager in our *EDOM* is conducive to deciding the number of cores needed to optimize the energy efficiency of multicore-based database systems where main memory becomes a scarce resource.

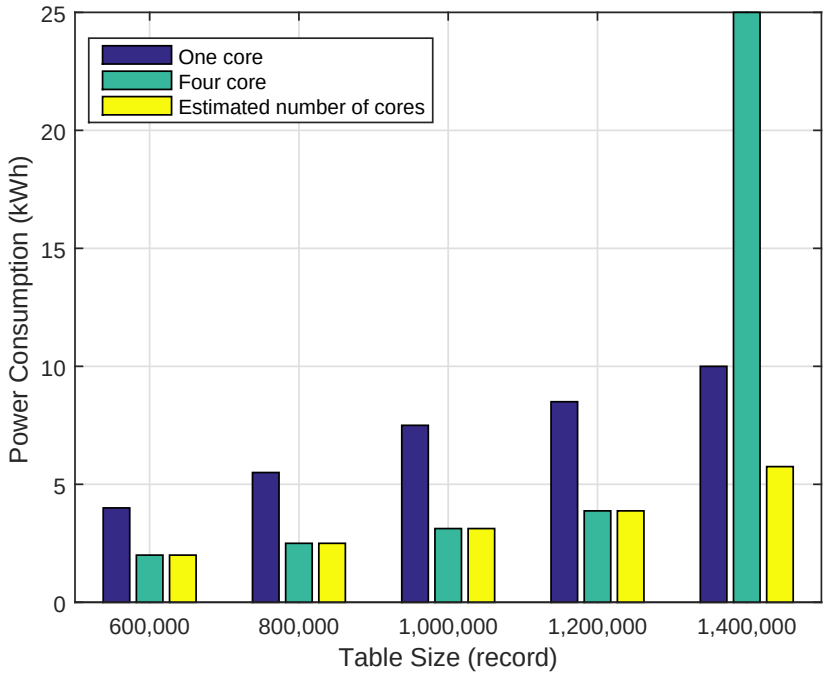


Figure 3.4: The validation of energy consumption governed by our multicore manager.

The testbed used in our experiments is equipped with a Celeron(R) 2.2 GHz CPU, 1.0 GBytes RAM, and a 160 GBytes SATA disk. Temperature sensors and watchdog are applied to monitor the disk, inlet and outlet temperatures. The configuration parameters are summarized in Table 3.3. The ambient temperature is set to 23.2 °C .

Table 3.3 Testbed Configurations

Hardware	Software
1 × Intel(R) Celeron(R) 450@2.2GHz 1 × 1.0 GBytes of RAM 1 × WD 500 GBytes Sata disk (WD5000AAKS-75M0A0 [1])	Ubuntu 10.04 Linux kernel 2.6.32

3.4 Experimental Results

We have conducted extensive experiments to demonstrate the usage and effectiveness of EDOM. We apply the developed EDOM to evaluate the energy efficiency and performance of a multicore-based database server system.

In this part of the study, we first investigate the impacts of CPU utilization and multicore on the database system. Then, we compare the energy efficiency of different database operations. Finally, we demonstrate how the indexing technique affects power consumption and performance of the database system.

3.4.1 CPU Utilization and Mulitcores

We conduct extensive experiments to capture the power consumption characteristics of CPU utilization and multicores.

In the first group of experiments, we focus on the impacts of CPU utilization and multicores on the energy efficiency and performance of the tested database system. To make fair comparison, we keep the number of queries executed by the system a constant under various hardware configurations. By doing so, we demonstrate how multicores under a wide range of configurations affect the database system.

Energy Efficiency of the Outer-Join Operation

In this experiment, we issue a fixed number of outer-join queries while changing the database table sizes and number of cores in the system. Fig. 3.5(a) reveals that regardless of the number of cores, energy consumption of the database system goes up when the table size increases. This trend is reasonable, because the query processing time is enlarged when the data volume increases with the increasing table size. The large processing time gives rise to the high energy consumption.

Now we compare the three curves plotted in Fig. 3.5(a). When the table size is smaller than 1.4×10^6 records, increasing the number of cores in the database system significantly reduces the energy consumption caused by processing the outer-join queries. When we add extra cores into the database system, the query processing time is noticeably shortened, which in turn conserves energy.

Interestingly, the trend is inapplicable for cases where the table size becomes very large. For example, the energy consumption of the four-core system is much larger than the two-core counterpart when the table size is 1.6×10^6 records. The four-core system is unable to conserve energy under the large-table-size condition, because the main memory requirement imposed by the four cores exceed the available memory (i.e., 4GB) in the tested system. We conclude that increasing the number of cores is an effective way to save energy of a database system, provided that the system's main memory resource can meet the multicore system's needs.

Fig. 3.5 (b)indicates that given a fixed amount of outer-join queries, executed under different CPU utilization within one core, the most energy efficient condition is the 100% CPU utilization. And as the table size increases, the more CPU utilization it takes the slower the energy consumption grows. The energy consumption of 100% CPU utilization at the point of table size 2million is even 37% of the energy consumption of 25% CPU utilization. The reason is the less CPU utilization used, the more idle status consumption is taken into account of the whole energy consumption.

We are in a position to evaluate the impacts of CPU utilization on the energy consumption of the database system processing outer-join queries. We test a total of four cases, in each of

which the CPU utilization is fixed.Because the focus of this experiment is CPU utilization, we set the number of cores to one, avoiding any side effect incurred by the multicores.

The experimental results demonstrate that the most energy-efficient case is the one when the CPU utilization is set to 100%. For instance, when the table size is configured to 2.0×10^6 records, the 100%-CPU-utilization case reduces the energy consumption of the 25%-CPU-utilization case by more than 63%. Such a significant energy saving is expected, and the reason is two-fold. First, the 25%-CPU-utilization case exhibits a large number of small idling time periods. Second, the database system is unable to keep the CPU in the low-power mode to conserve energy during these short idling time intervals.

We also observe that regardless of the CPU utilization value, a large table size leads to high energy consumption. This observation is consistent with that drawn from Fig. 3.5(a).

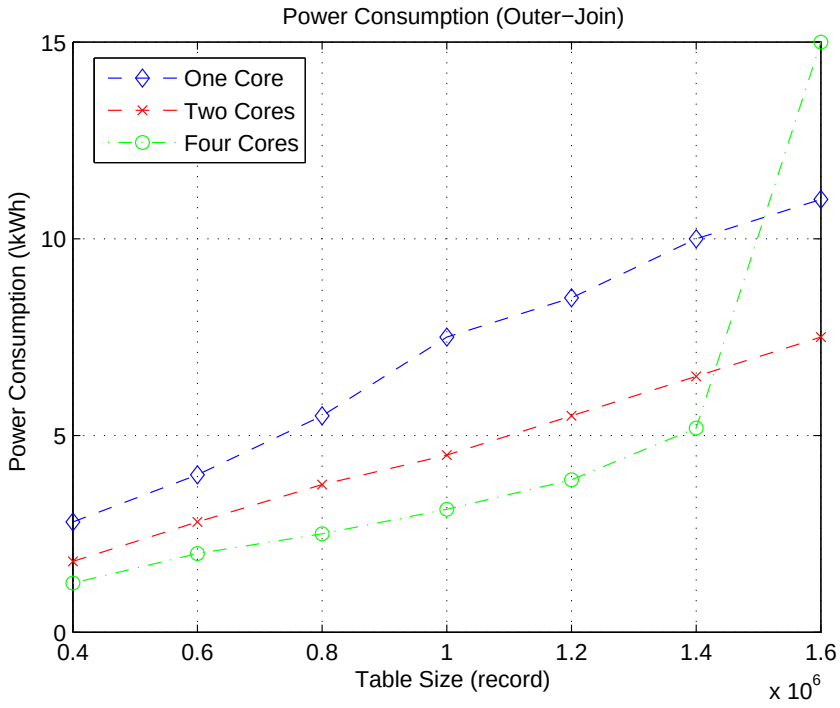
Energy Efficiency of the Cross-Join Operation

In this set of experiments, we evaluate the energy efficiency of the database system using cross-join queries. Similar to the previous experiments discussed in Section 3.4.1, in this group of experiments a fixed number of cross-join queries are executed while varying the table size and number of multicores.

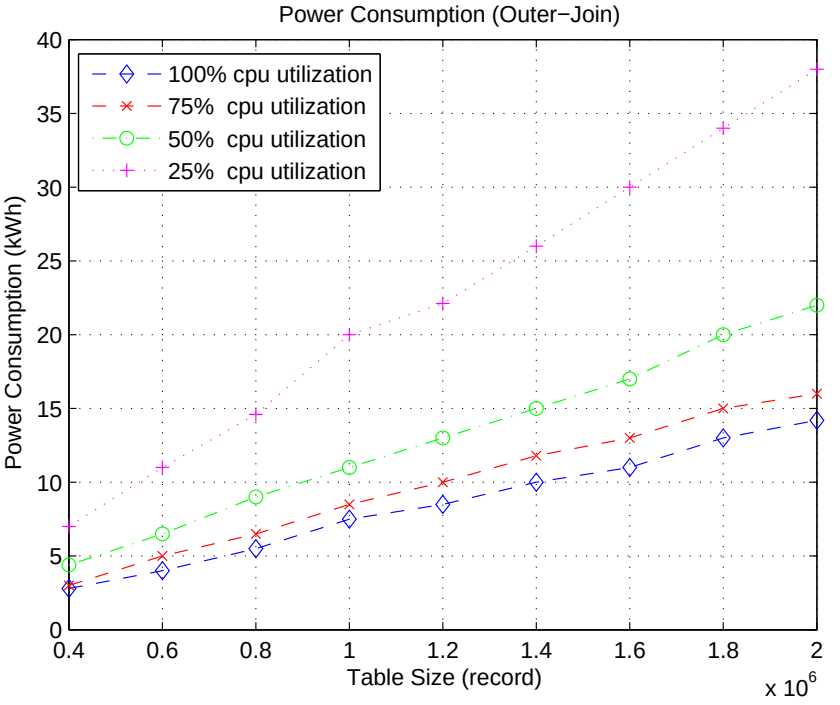
We observe that various types of database operations have different impacts on energy efficiency. Nevertheless, the power consumption trend shown in Fig. 3.6(a) is similar to that of Fig. 3.5(a); thus, regardless of the number of cores, energy consumption of the database system goes up when the table size increases. This is because increasing table size enlarges data volume and its processing time, which in turn consume more energy.

The comparison of the two curves in Fig. 3.6(a) reveals that in the case where the table size is set to 1800, an extra core dramatically reduces the energy consumption. This observation is consistent with the one drawn from Fig. 3.5(a).

The energy consumption of the three-core and four-core cases are not plotted in Fig. 3.6(a), because the system's main memory is so heavily utilized that multiple cores are unable to process any query. For example, the three-core system exhibits excessive long response times even when the table size is as small as 1000. We conclude that the memory resource becomes



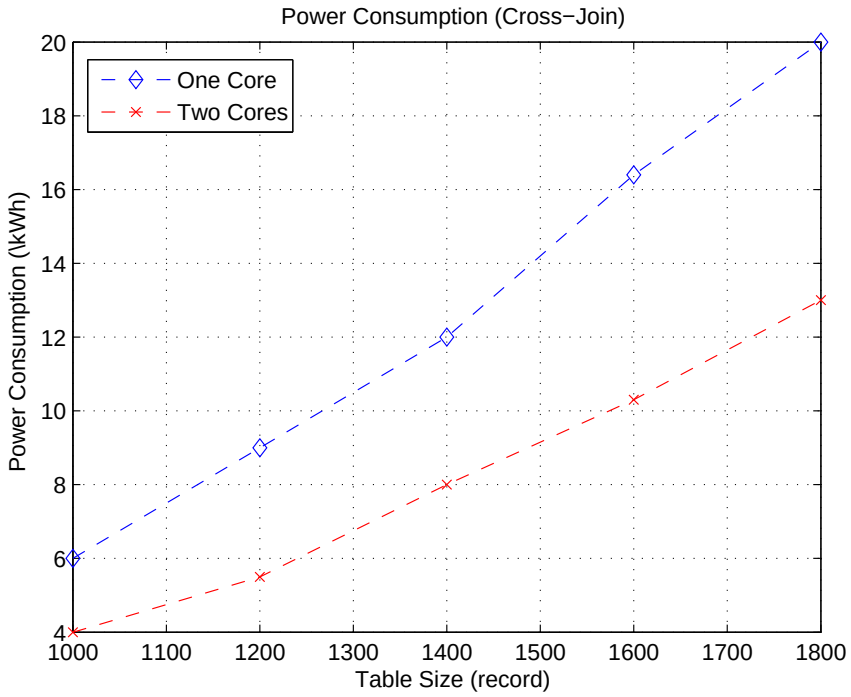
(a) Power Consumption with Multicores



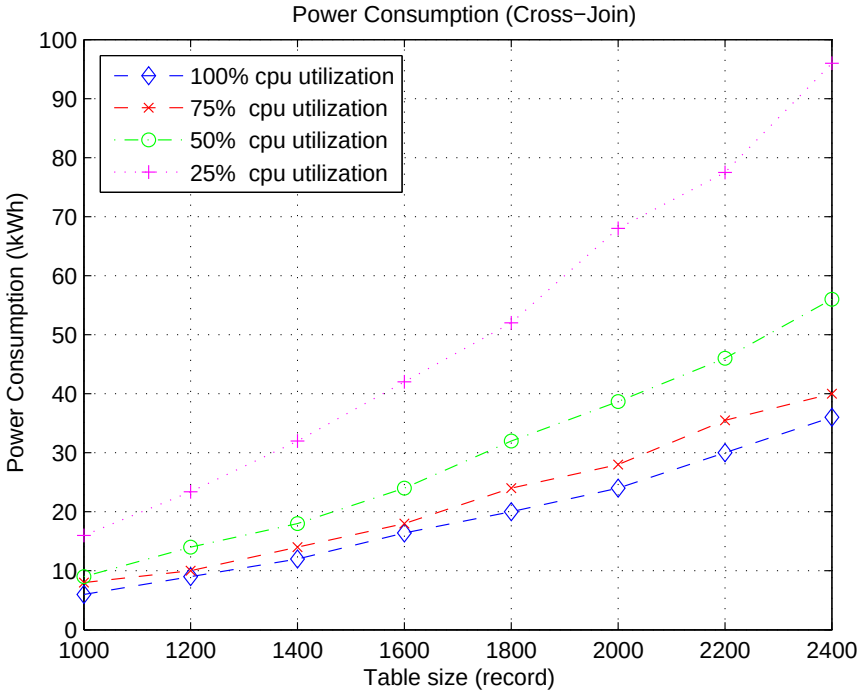
(b) Power Consumption under different CPU utilization

Figure 3.5: Power consumption profiling of query: Outer-Join

a performance bottleneck of the multicore-based database system. Moreover, compared with



(a) Power Consumption with Multicores



(b) Power Consumption under different CPU utilization

Figure 3.6: Power consumption profiling of query: Cross-Join

the outer-join operations, the energy behavior of cross-join queries are more sensitive to main memory capacity of the system.

Fig. 3.6(b) shares similar energy consumption patterns as those of Fig. 3.5(b) (Section 3.4.1). Thus, the system's energy efficiency can be significantly improved by pushing CPU utilization up to 100%.

Fig. 3.6(b) confirms a trend illustrated in Fig. 3.6(a) that increasing table size leads to a high energy-consumption level. Compared with the 100%-CPU-utilization case, the energy consumption of the 25%-CPU-utilization case is more sensitive to the table size. In other words, when we enlarge the table size in the 25%-CPU-utilization case, the system's energy consumption increases faster than the 100%-CPU-utilization case.

If the table size is smaller than 1600, increasing the CPU utilization from 75% up to 100% has a marginal energy-efficiency improvement. On the other hand, when it comes to a large table size (e.g., 1600-2400), a CPU-utilization increase of 25% up from 75% makes a noticeable reduction in power consumption.

An insightful conclusion drawn from this group of experiments is that we can improve the energy efficiency of database systems by making a full usage of multicore processors in servers.

Performance of the Outer-Join Operation

Now we study performance of the outer-join operations. Fig. 3.8(a) shows the performance as a function of the number of table size in the one-core, two-core, and four-core cases. The results reveal that the outer-join operations exhibit better performance in the four-core configuration than in the other two cases. The four cores reduce the execution time of performing the outer-join queries, thereby improving system energy efficiency (see also Fig. 3.5(a)). We observe that in the four-core case, the execution time sharply climbs up when the table size exceeds 1.4 million. Such a performance degradation is attributed to the problem that the main memory capacity is unable to meet the needs of the large table size.

Fig. 3.8(b) illustrates the impact of CPU utilization on the performance of the system running outer-join operations. In this group of experiments, we vary the table size from 0.02 to 2.0 million; we also tested four cases where the CPU utilization is kept at 100%, 75%, 50%, and 25%, respectively. Fig. 3.8(b) shows that increasing CPU utilization shortens the

time spent in performing the outer-join operations. This performance trend becomes more pronounced when the table size is large. The 100%-CPU-utilization case outperforms the other three cases, because CPU idle times in the other three scenarios slow down the query process performance.

We investigate the memory usage under various table sizes and number of cores. Fig. 3.8(a) indicates that regardless of the number of cores, increasing the table size slightly drives the memory usage up. Compared with the memory usage in the one-core and two-core systems, memory usage of the four-core system is more sensitive to the page size. For example, when we increase the table size from 0.4 to 1.2 millions, the memory usage of the four-core system increases from 63% to 92%, whereas the memory usage of the single core system only slightly goes up to 26% from 15%. The four-core system's query processing performance is significantly deteriorated when the memory usage is very high, which represents a high demand on memory resources.

Fig. 3.8(b) reveals the impact of CPU utilization on system memory usage. The experimental results suggest that the CPU utilization has no noticeable impact on memory usage. We conclude that a database system's memory usage largely depends on the table size and the number of cores in the system.

Performance of the Cross-Join Operation

In this group of experiments, we evaluate the performance of cross-join operations running on multi-core systems. Fig. 4.4(a) shows a similar performance trend as that plotted in Fig. 3.8(a). We only show the results of the single-core and two-core cases, because the query processing time of the four-core system is extremely long due to the high memory usage. The execution time of the cross-join operations is significantly reduced by adding an extra core into the database system. In addition, the execution time of the two-core system is less sensitive to the table size than that of the single-core system, thanks to high performance offered by the two cores.

Fig. 4.4(b) shows the impact of CPU utilization on the cross-join performance. A high CPU utilization helps in boosting the processing performance of cross-join queries. This performance trend is consistent with that observed from Fig. 3.8(b).

Fig. 4.4(c) reveals the memory usage of cross-join operations under various table size and the number of cores. Fig. 4.4(c) shows that the memory usage increases almost linearly with the increasing table size. We observe that compared with the memory usage of the outer-join operations, the memory usage of cross-join operations is more sensitive to the table size (see also Fig. 3.8(c) and Fig. 4.4(c)). More detailed comparison between outer-join and cross-join operations can be found in Section 3.4.2.

3.4.2 Outer-Join vs. Cross-Join Operations

Now we compare the energy behaviors between outer-join and cross-join operations in multicore database systems.

Impact of multicores on outer-join and cross-join queries

We pay particular attention to the impact of multicores on the outer-join and cross-join queries under the changing table size. The results plotted in Figs. 3.10(a) and (b) indicate that the query types have significant impacts on energy efficiency. For example, the energy consumption of outer-join is only 0.047% and 0.078% of that of cross-join when table size is set to 2400 and 1800, respectively (see Fig. 3.10(a)) and Fig. 3.10(b)). Thus, the energy consumption of cross-join queries is 1000-10000 times higher than that of outer-join queries. This energy consumption trend is attributed to two reasons. First, cross-join queries give rise to a huge amount of data loaded from disks into main memory. Second, the database system allocates a significant portion of CPU resources to process cross-join queries.

We observe from Fig. 3.10(a) that in the single-core case, outer-join queries are less sensitive to table size than cross-join queries. Compared with outer-join operations, cross-join's energy consumption can be substantially reduced by applying optimization algorithms (e.g.,

relational optimizer) to maintain small query sizes in multicore-based database systems. Unlike cross-join queries, outer-join queries may enjoy marginal benefit from the optimization algorithms.

Interestingly, Fig. 3.10(b) shows that in the two-core case, cross-join queries become less sensitive to table size than outer-join queries. Nevertheless, the wide energy-consumption gap between outer-join and cross-join is alleviated by increasing the number of cores from one to two. There is no doubt that employing multiple cores and reducing query size are two efficient ways of narrowing the gap between the outer-join and cross join operations.

Impact of CPU utilization on outer-join and cross-join queries

Now we compare the difference between the outer-join and cross join queries from the perspectives of CPU utilization impact on energy consumption. Again, we increase the table size from $1,000$ to $2,400$ records with an increment of 200. Fig. 3.11 and Fig. 3.12 reveals that regardless of CPU utilization, the cross-join query is a whole lot more energy expensive than the outer-join one. This trend is consistent with the results plotted in Fig. 3.10.

An intriguing observation drawn from Fig. 3.11 is that the cross-join operation's energy-consumption increasing ratio is more sensitive to CPU utilization and table size than that of the outer-join one. For example, let us consider a scenario where the table size is gradually increased from $1,000$ to $2,400$. In the 25%-CPU-utilization case, the energy consumption of outer-join query increased by approximately 50%; in the 100%-CPU-utilization case, the outer-join's energy consumption is increased by more than 122%. In the 25%-CPU-utilization and 100%-CPU-utilization cases, the cross-join operation's energy consumption is increased by 308% and 337%, respectively.

The implication of the results shown in Fig. 3.11 is that under heavy CPU utilization, reducing table size becomes a feasible approach to noticeably conserving energy consumption of the cross-join operation.

Energy efficiency vs. performance impacts

In this group of experiments, we compare the performance in terms of response time between outer-join and cross-join queries under various CPU workload. The performance trend observed in this set of experiments may shed some light on the energy-consumption comparisons between the two query types (see Sections 3.4.2 and 3.4.2).

Like the configuration of the previous experiment, the table size is increased from $1,000$ to $2,400$ records with an increment of 200; the CPU utilization is set to 25%, 50%, 75%, and 100%, respectively. Fig. 3.13 and Fig. 3.14 shows that the cross-join query's response time is almost 222 times longer than that of the outer-join one when the table size and CPU utilization are set to $1,600$ records and 75%, respectively.

Not surprisingly, the performance trends revealed in Fig. 3.13 are similar to the energy-efficiency trends observed in Fig. 3.13. We conclude that the energy consumption of the two query types are strongly correlated to their response time. The experimental results suggest that any algorithm aiming to shorten the response times of the queries is likely to improve the energy efficiency of the queries running in multicore systems.

The results from this group of experiments also confirm that under high CPU workload (see, for example, Fig. 3.14(b)), reducing table size can significantly shorten the response times of the queries. This conclusion is especially true for the outer-join operation.

Fig. 3.15 shows the performance comparisons between the outer-join and cross-join queries under the single-core and double-core cases. Very interestingly, we observe that the speedup efficiency of cross-join is higher than that of outer-join. For example, when the table size is set to 1000, the speedups of cross-join and outer-join are 1.92 and 1.51. Overall, the speedup efficiency of both outer-join and cross-join is improved with the increasing table size.

Fig. 3.16 illustrates the comparisons between outer-join and cross-join operations from the perspective of memory usage. The results show that compared with outer-join's memory usage, cross-join's memory usage is more sensitive to table size. Such a trend becomes more pronounced when we increase the number of cores from one (see Fig. 3.16(a)) to two (see Fig. 3.16(b)). For instance, Fig. 3.16(b) reveals that the memory usage of cross-join goes

up from 42% to almost 100% when the table size is increased from 1000 to 1800, whereas the outer-join operation's memory usage stays fairly flat regardless of the table size and the number of cores.

3.4.3 The Indexing Technique

Now we investigate the indexing technique's impacts on the energy behaviors of the outer-join queries. We only demonstrate the power consumption of outer-join, because cross-join's power consumption has a similar trend. In this group of experiments, we vary the table size from 1.0×10^6 to 2.0×10^6.

Fig. 3.17 intuitively shows that indexing substantially affects the outer-join operation's power consumption, performance, and memory usage. The energy trend plotted in Fig. 3.17(a) is similar to the performance trend illustrated in Fig. 3.17(b), implying that the performance and energy efficiency of outer-join have a tight correlation. The experimental results suggest that when it comes to indexing, there is no need to make tradeoff between energy efficiency and performance.

Figs. 3.17(a) and 3.17(b) indicate that indexing not only boosts outer-join performance, but also makes outer-join more energy efficient. The energy efficiency and performance improvements offered by indexing become more significant when the table size is growing up. For example, when the table size is small, the indexing scheme has a limited impact on power consumption; indexing only manages to reduce the power consumption by 4.8%. If we change the table size to 1.4×10^6, indexing is able to offer an energy saving of 23.9%.

We observe from Fig. 3.17(c) that the indexing technique significantly reduces the memory usage of the outer-join query. For example, when we set the table size to 1.0×10^6, the memory usage rate of the indexing case is 69.3%; without indexing, the memory usage rate goes up to 75.7%. The memory usage results show evidence that indexing improves outer-join's performance by alleviating memory load in the multicore system. The indexing technique proactively reduces the amount of data loaded from the disks to the main memory, which in turn noticeably cuts the query response time. We conclude that with indexing in place, the outer-join queries

are processed in an energy efficient way thanks to the shortened query response times made possible by indices.

After evaluating the energy overhead incurred by creating indices, we reach a conclusion that the energy overhead caused by indexing is trivial and; therefore, we ignored the energy overhead results from the figures.

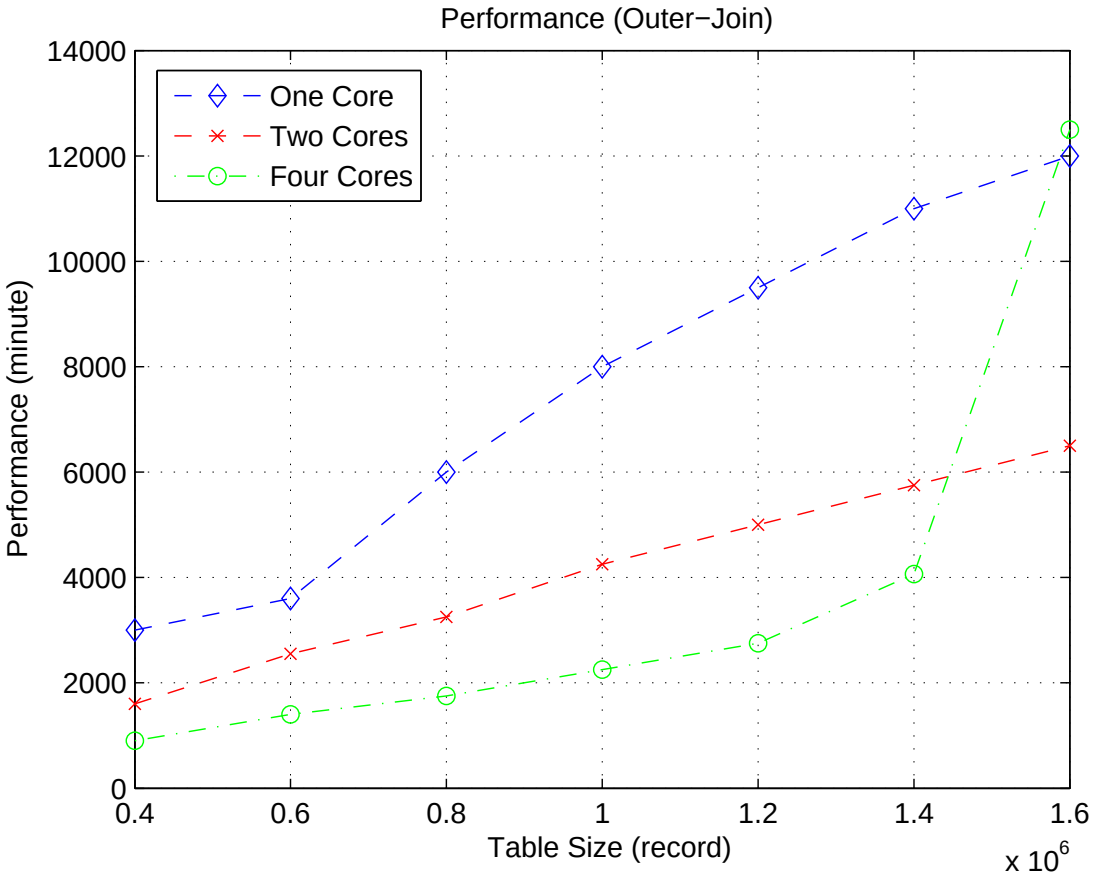
3.5 Summary

We started this chapter by investigating the workload conditions and proposing metrics as well as the guidelines of energy-efficiency benchmarks. Then, we proposed *EDOM* - a tool systematically evaluating and optimizing the energy-efficiency of multicore-based database systems.

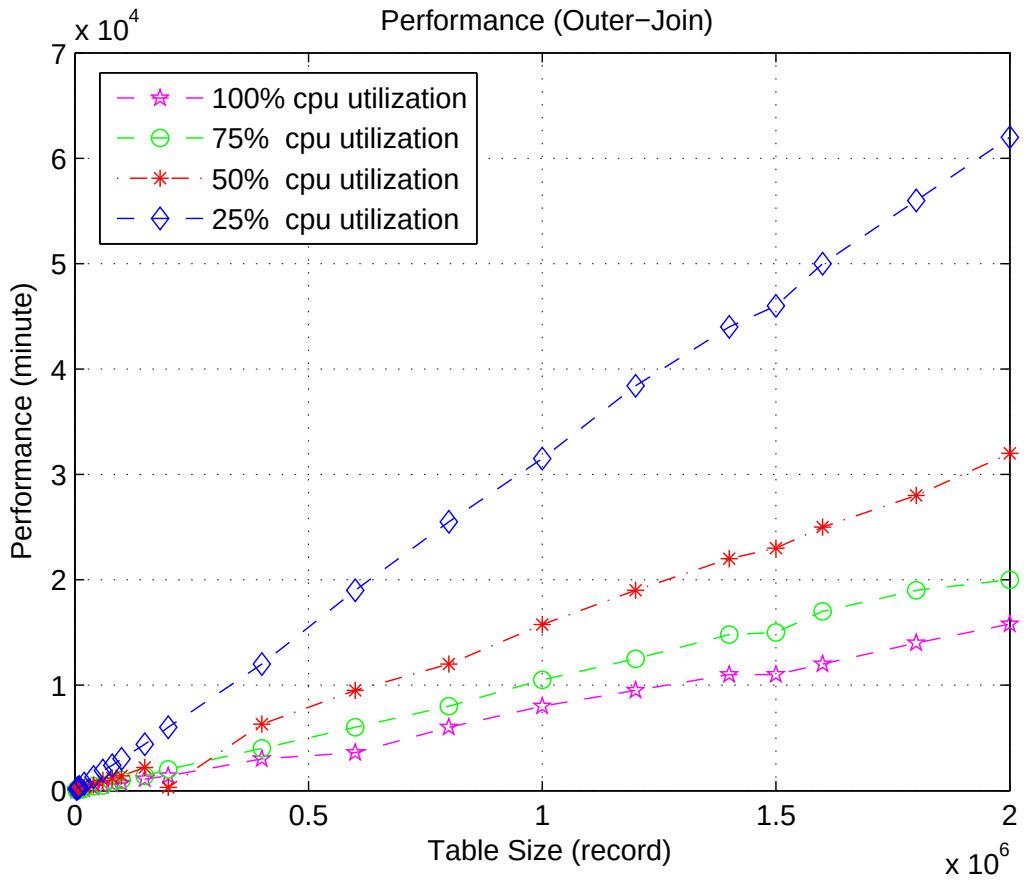
We incorporated the TPC-W benchmark database in *EDOM* to resemble real-world database systems. The *EDOM* tool employs the PostgreSQL database to evaluate the energy efficiency of two database queries, namely, outer-join and cross-join operations. *EDOM* offers a simple yet efficient way of measuring energy efficiency of database queries running on multicore processors; *EDOM* shows the correlation between CPU utilization and energy efficiency.

At the heart of *EDOM* is a multicore manager making a good tradeoff between energy efficiency and performance in database systems. *EDOM* leverages a memory usage model to estimate memory utilization using query types and database characteristics. *EDOM* alleviates the memory swapping problem by determining the most appropriate number of cores. We showed that *EDOM* substantially improves energy efficiency of multicore-based database systems by addressing the memory swapping issue.

Our experimental results and analysis indicate that our tool is a simple yet efficient platform to measure, improve, and optimize the queries, hardware configurations, and resource allocations multicore-based databases systems housed in data centers. One salient feature of *EDOM* lies in its high flexibility and adaptability, which allow *EDOM* to be customized and populated according to any research and application domain.

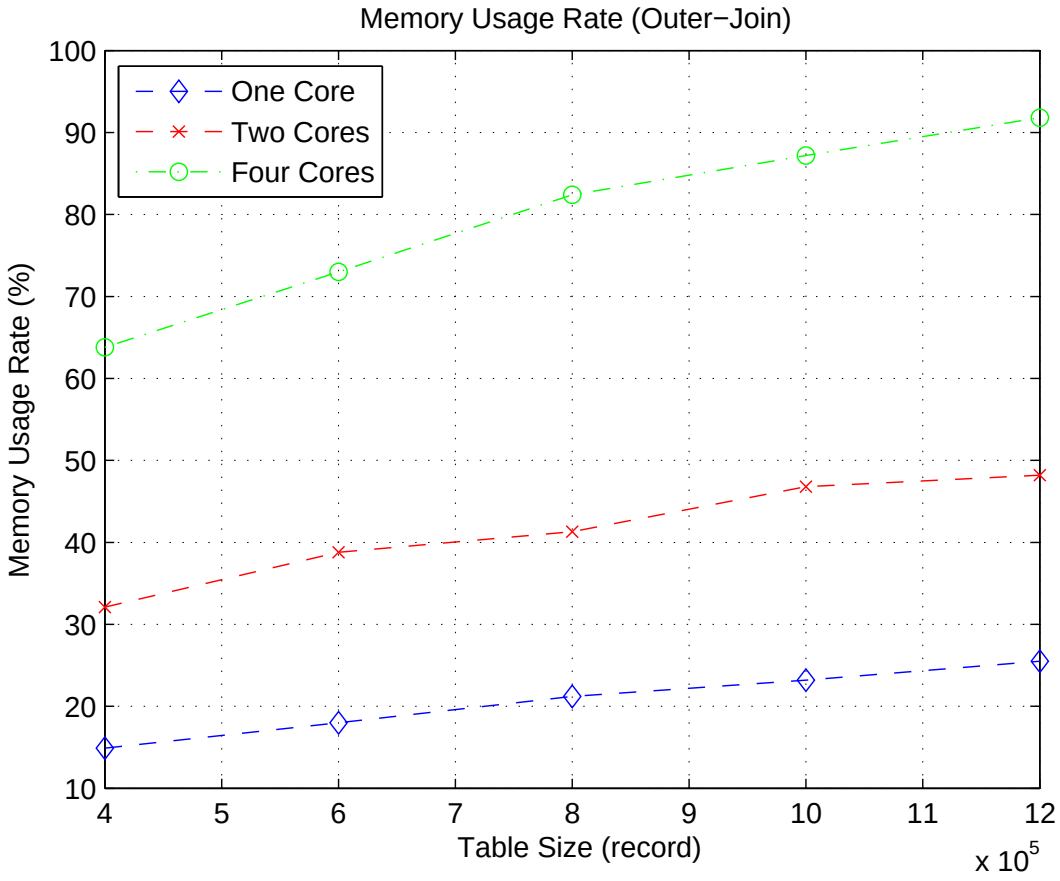


(a) Time Consumption with Multicores

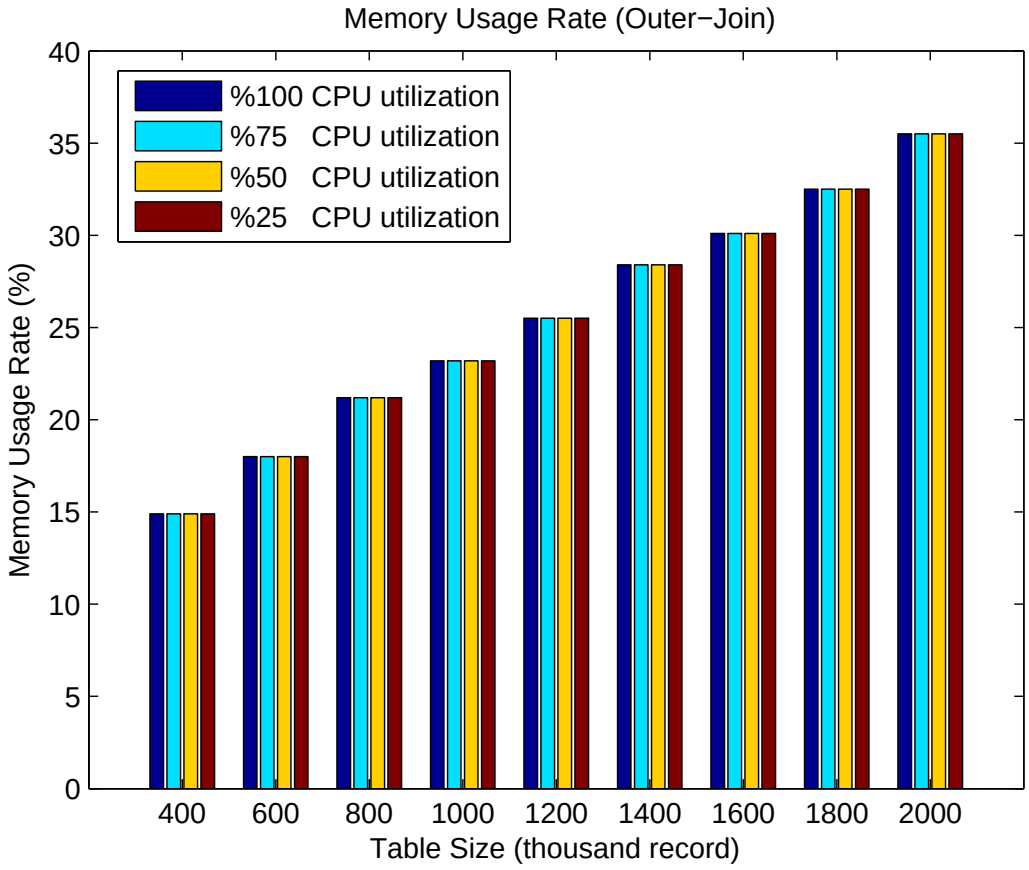


(b) Time Consumption under different CPU utilization

Figure 3.7: Performance profiling of Outer-Join under different CPU utilization and multicores

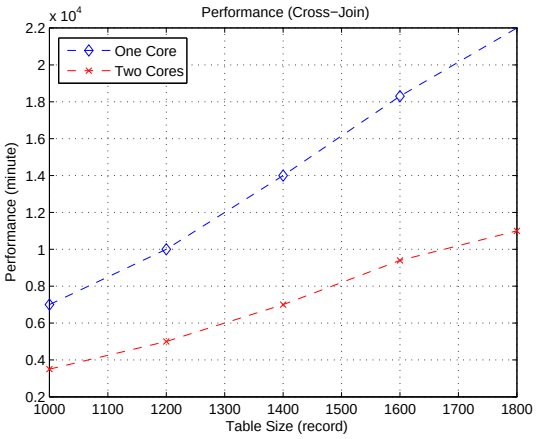


(a) Memory Usage under different CPU utilization

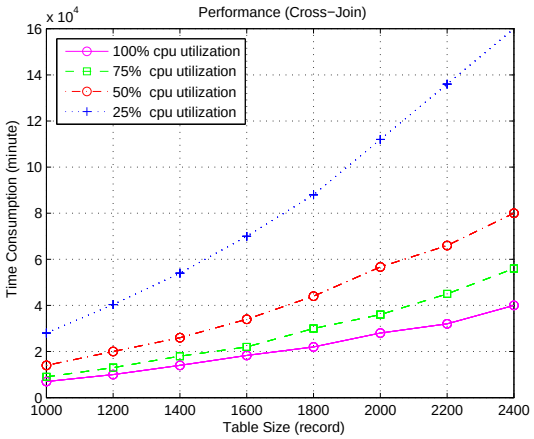


(b) Memory Usage under different CPU utilization

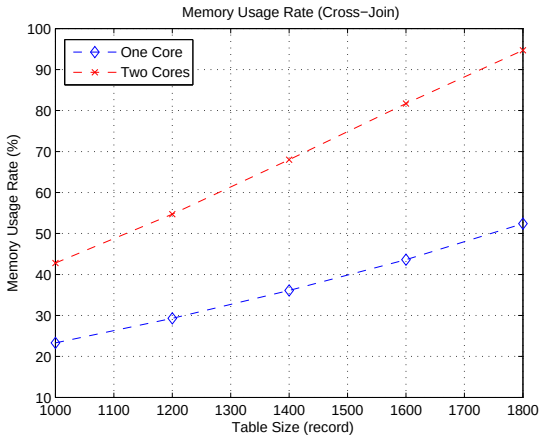
Figure 3.8: Performance profiling of Outer-Join under different CPU utilization and multicores



(a) Time Consumption with Multicores

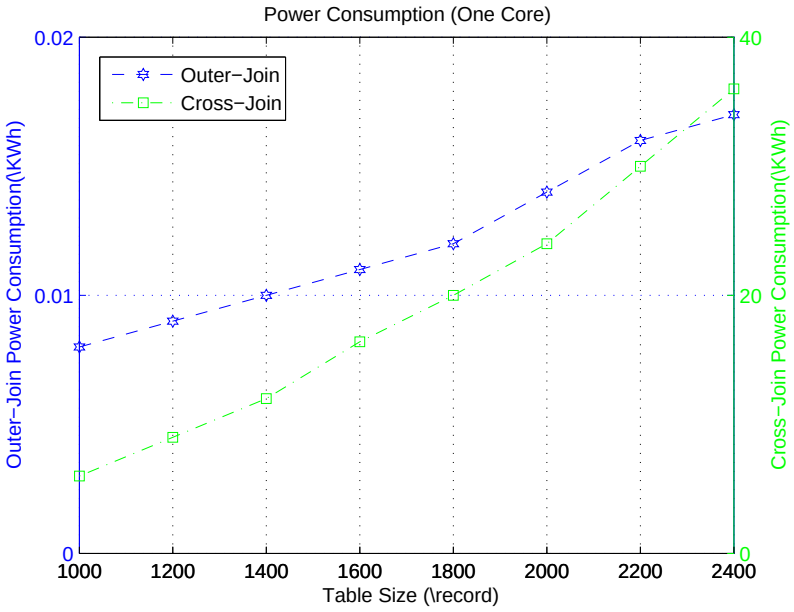


(b) Time Consumption under different CPU utilization

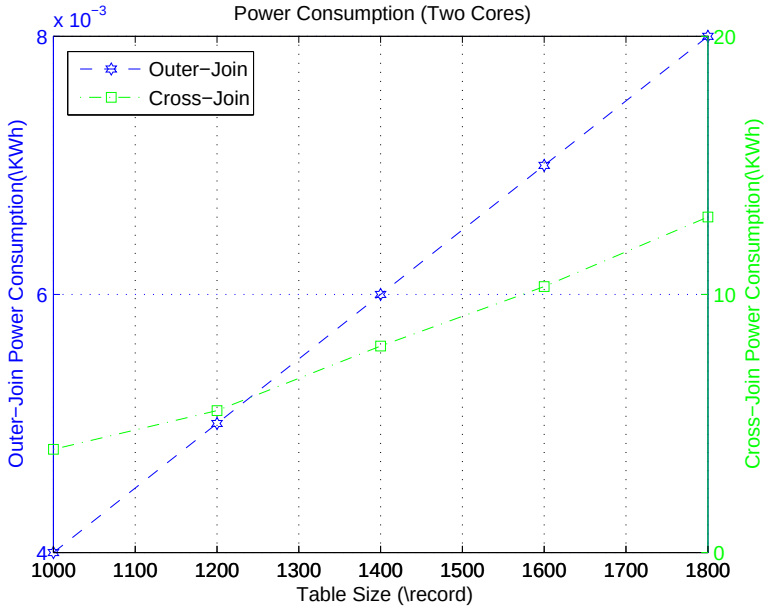


(c) Memory Usage under different CPU utilization

Figure 3.9: Performance profiling of Cross-Join under different CPU utilization and multicores

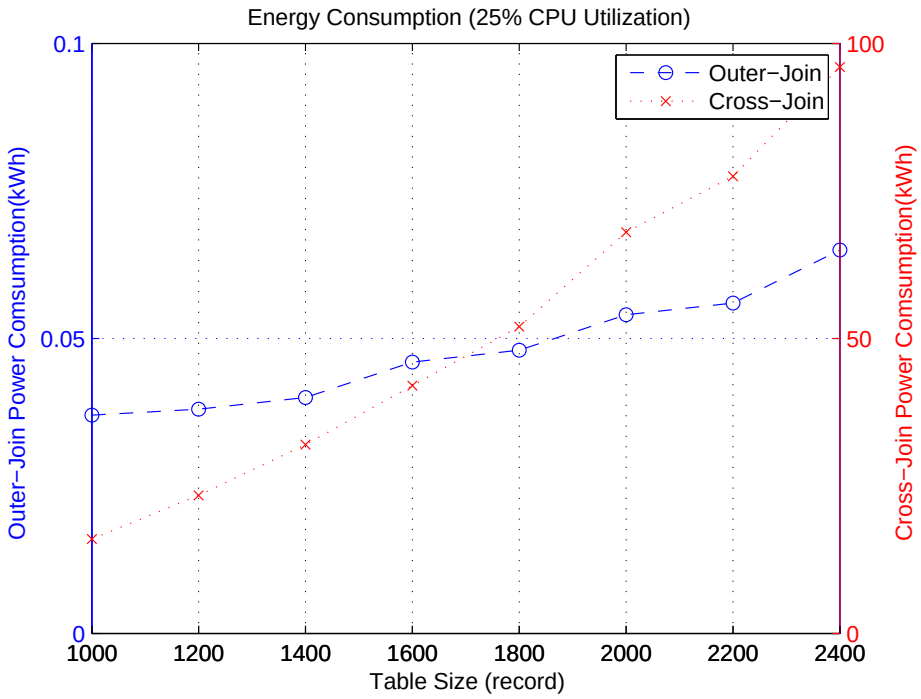


(a) Power consumption of outer-join and cross-join in the single-core case

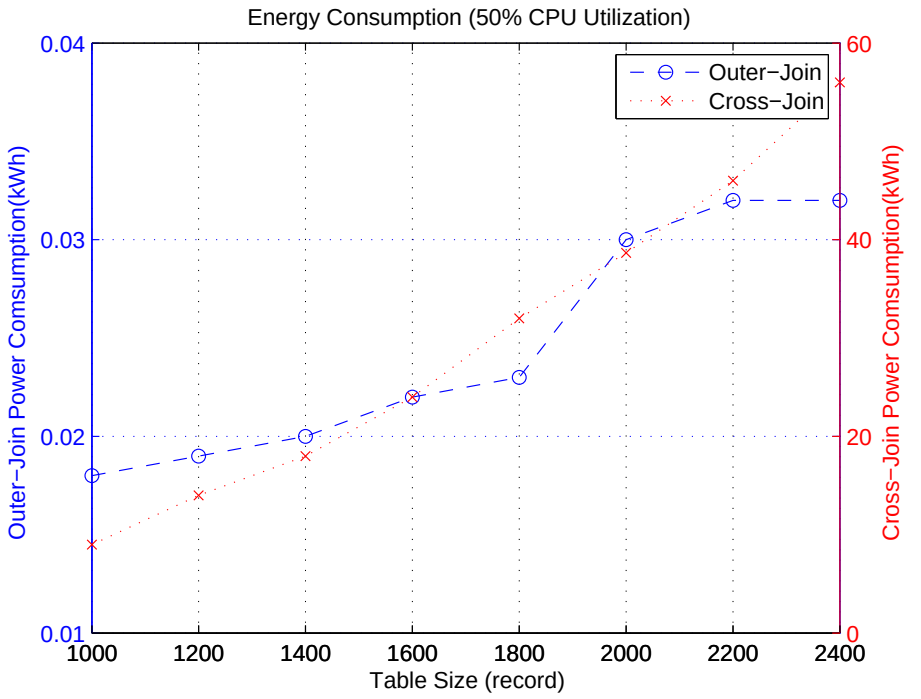


(b) Power consumption of outer-join and cross-join in the two-core case

Figure 3.10: Power consumption comparision between outer-join and cross-join in multicore systems.

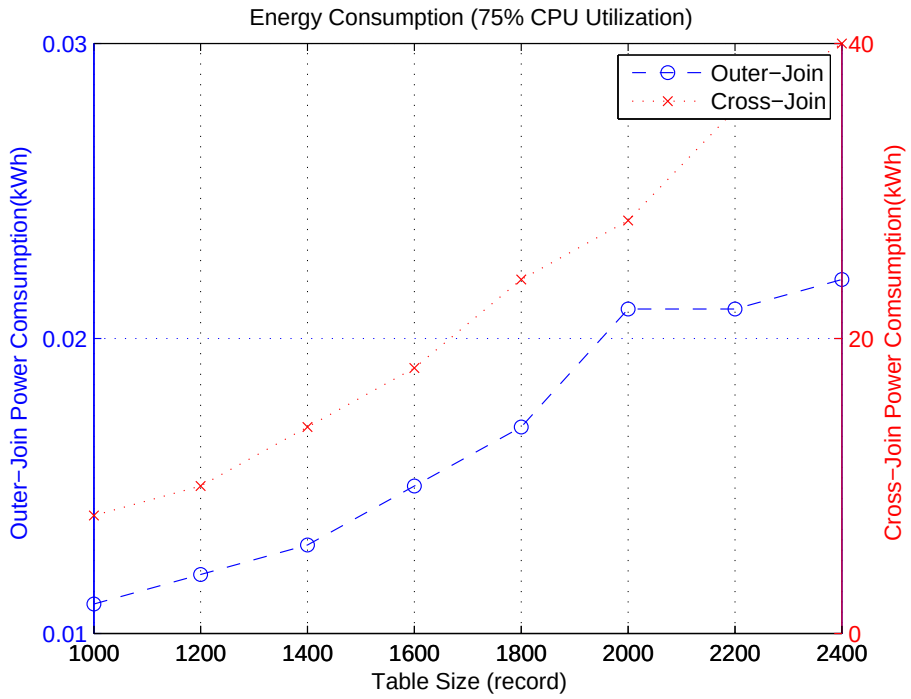


(a) Power consumption of the multicore system

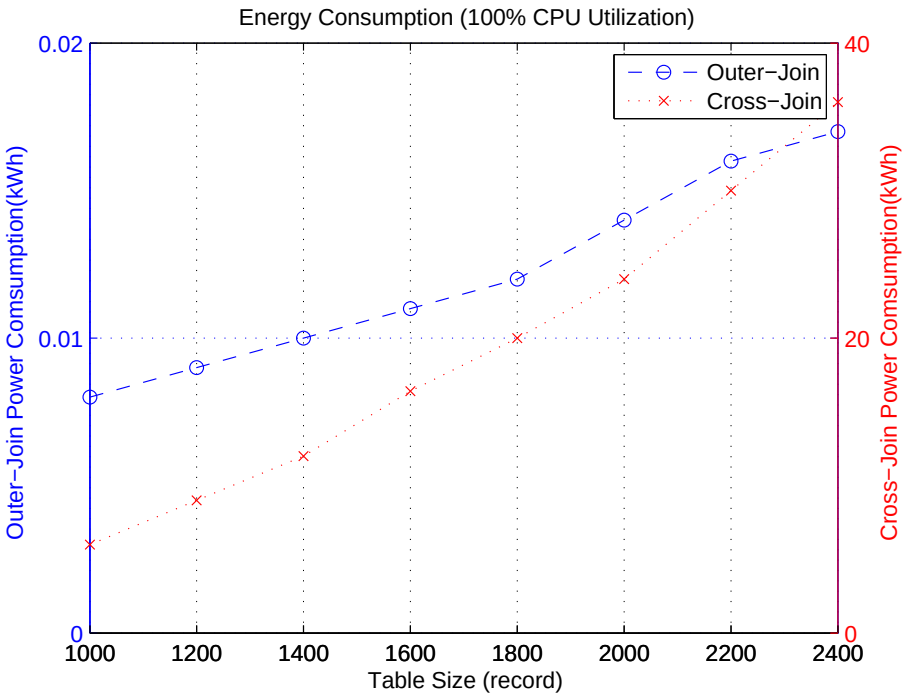


(b) Power consumption under different CPU utilization

Figure 3.11: Energy efficiency impact of outer-join and cross-join operations on multicore systems under various CPU utilization

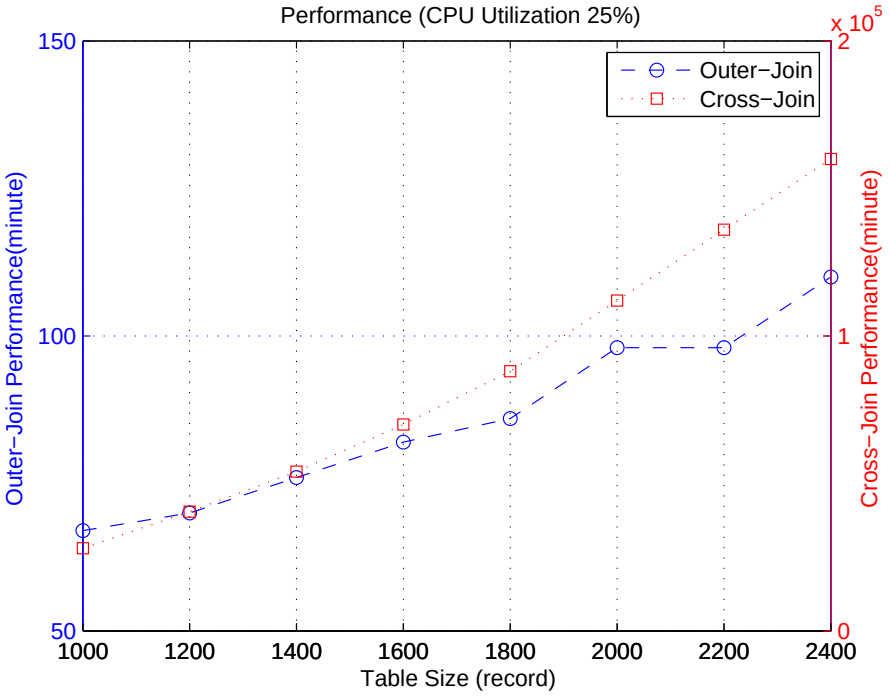


(a) Power consumption of the multicore system

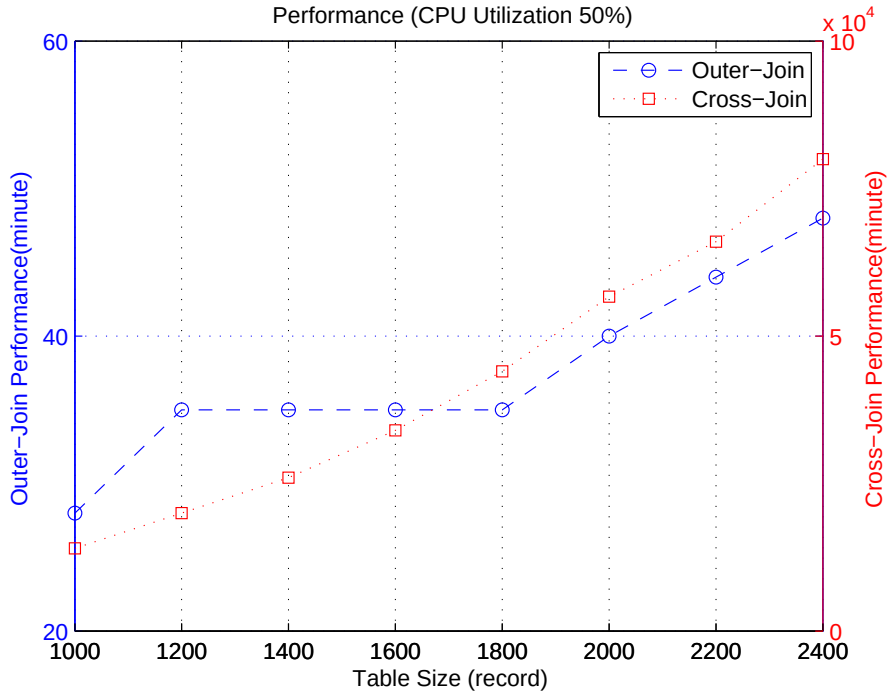


(b) Power consumption under different CPU utilization

Figure 3.12: Energy efficiency impact of outer-join and cross-join operations on multicore systems under various CPU utilization

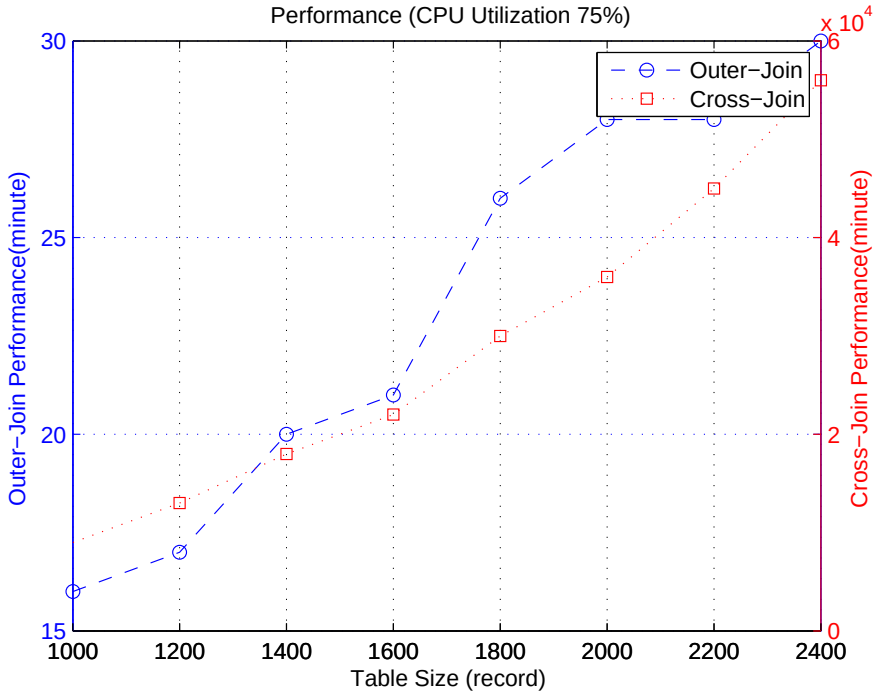


(a) Response time of outer-join and cross-join queries in the multicore system when CPU utilization is 25%

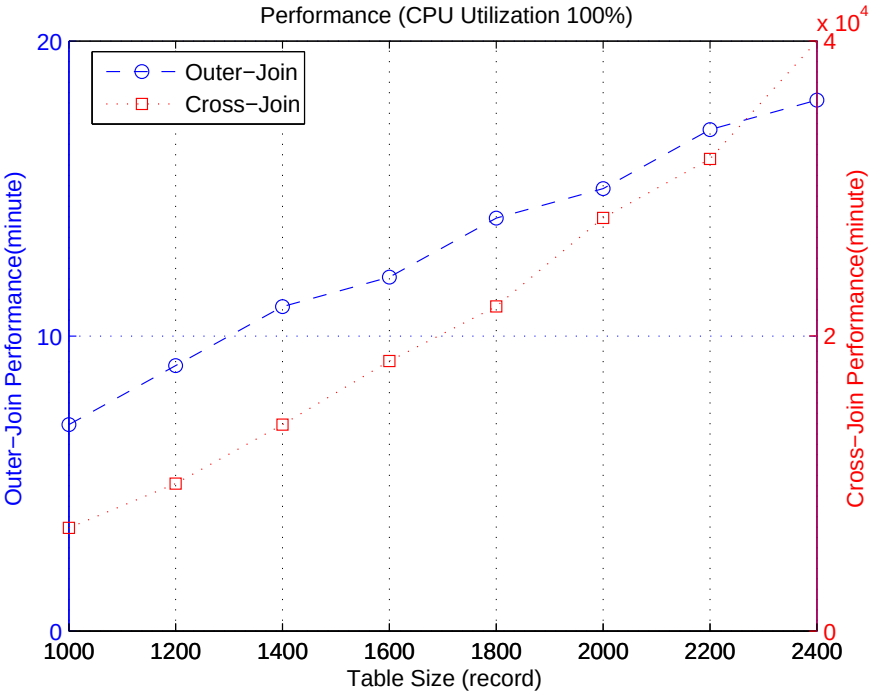


(b) Response time of outer-join and cross-join queries in the multicore system when CPU utilization is 50%

Figure 3.13: Performance comparision of outer-join and cross-join queries in the multicore system under various CPU utilization.

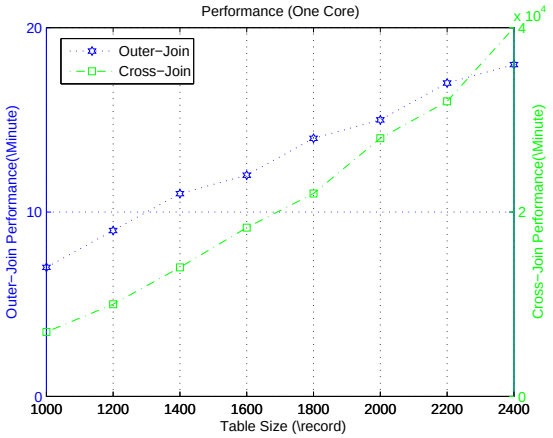


(a) Response time of outer-join and cross-join queries in the multicore system when CPU utilization is 75%

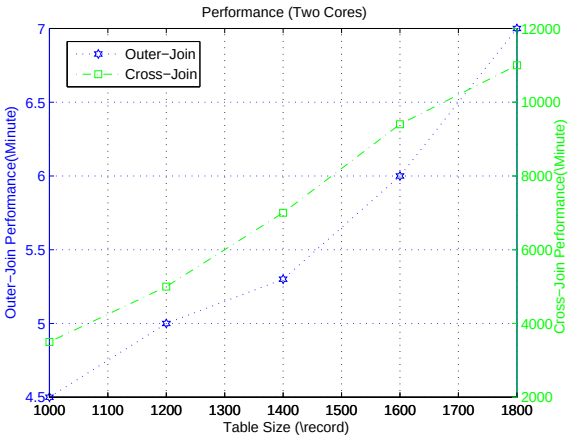


(b) Response time of outer-join and cross-join queries in the multicore system when CPU utilization is 100%

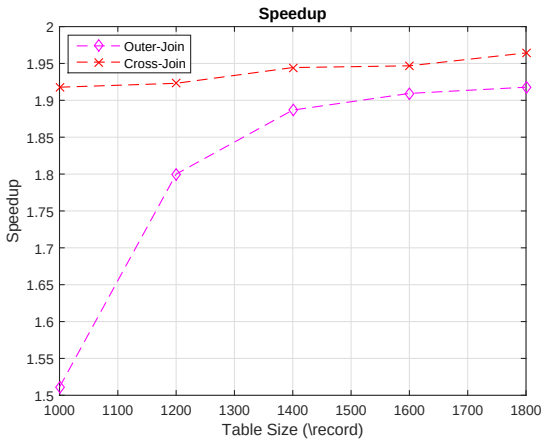
Figure 3.14: Performance comparision of outer-join and cross-join queries in the multicore system under various CPU utilization.



(a) Execution time comparison with one core

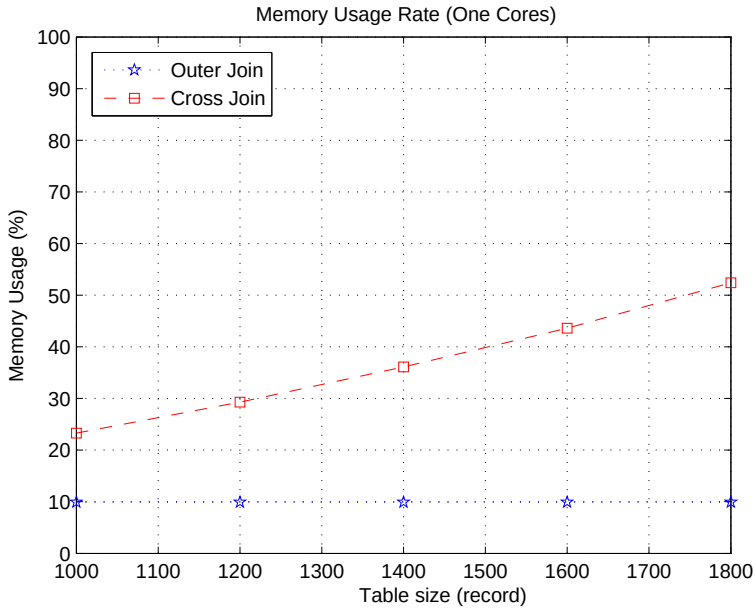


(b) Execution time comparison with two cores

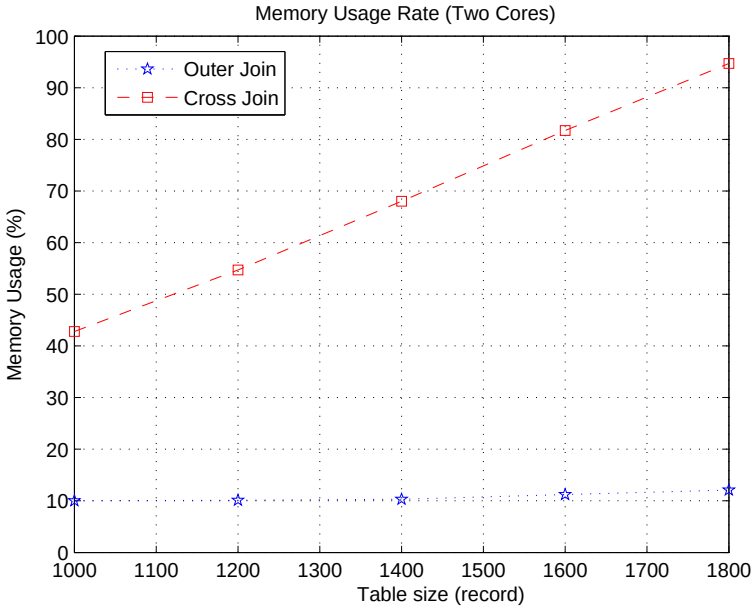


(c) Speedup comparison between outer-join and cross-join

Figure 3.15: Comparison of execution time between outer-join and cross-join under mutlticore situations

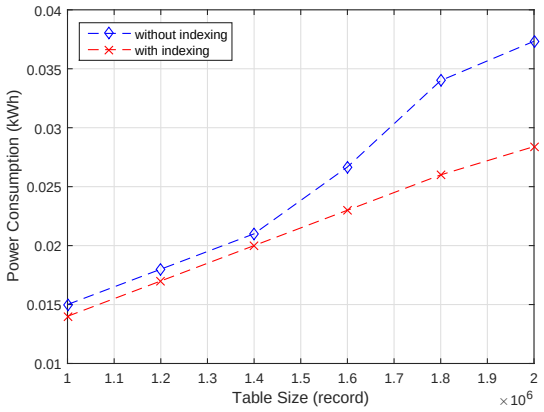


(a) Memory usage with Multicore

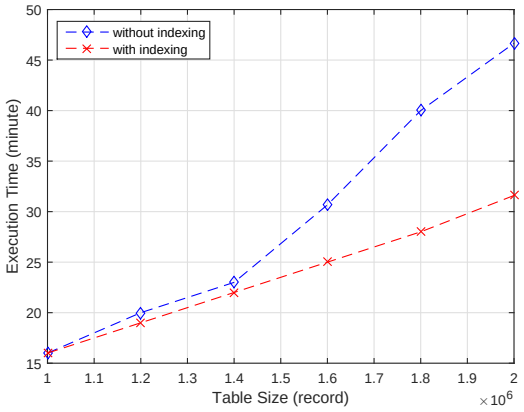


(b) Memory usage under different CPU utilization

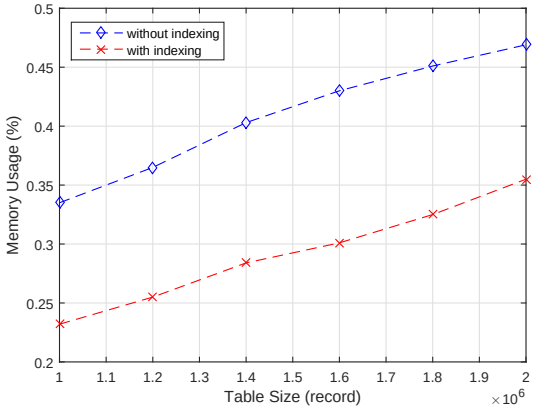
Figure 3.16: Impacts of outer-join and cross-join operations on the memory usage of multicore systems.



(a) Power consumption comparison with indexing



(b) Execution time comparison with indexing



(c) Memory usage comparison with indexing

Figure 3.17: Impacts of the indexing technique on energy efficiency of the outer-join and cross-join operations running in multicore systems.

Chapter 4

GreenDB

In this chapter, we propose an energy-efficient database system called *GreenDB* running on clusters. GreenDB applies a workload-skewness strategy by managing hot nodes coupled with a set of cold nodes in a database cluster.

GreenDB fetches popular data tables to hot nodes, aiming to keep cold nodes in the low-power mode in increased time periods. GreenDB is conducive to reducing the number of power-state transitions, thereby lowering energy-saving overhead. A prefetching model and an energy saving model are seamlessly integrated into GreenDB to facilitate the power management in database clusters. We quantitatively evaluate GreenDB's energy efficiency in terms of managing, fetching, and storing data. We compare GreenDB's prefetching strategy with the one implemented in Postgresql.

The findings show that the energy efficiency of GreenDB can be optimized by tuning system parameters, including table size, hit rates, number of nodes, number of disks, and inter-arrival delays.

The rest of the paper is organized as follows. The design and implementation of energy-efficiency *GreenDB* are discussed in Section 4.1. An energy-saving prediction model in GreenDB is presented in Section 4.2. Section 4.3 validates the energy efficiency of GreenDB by a set of experiments. Finally, Section 4.4 shows conclusions including the contributions of this research along with our future research directions.

4.1 System Design

4.1.1 Basic Ideas

A database cluster governed by *GreenDB* is divided into a group of cold nodes and a group of hot nodes. The overarching goal is to keep the cold nodes in the low-power state (e.g. shutting down) for a long time period, while making the hot nodes respond queries accessing popular data tables. There are three motivations behind partitioning nodes into the hot and cold groups.

• First, our preliminary findings suggest that improving the utilization of an active database node leads to high energy efficiency.

• Second, placing an entire node into the low-power state is more energy efficient than turning off disks in the node.

• Third, reducing the number of power-state transitions can lower the power management overhead.

4.1.2 Software Architecture

We design the software architecture of *GreenDB* (see Fig. 4.1, which manages a master node and a group of hot and cold database nodes (or nodes for short). To achieve high performance of database clusters, GreenDB proactively maintains a group of hot nodes that are always in the active mode. Our preliminary findings suggest that high node utilization improves energy efficiency; CPUs are a dominate contributor to the energy consumption of database applications. Therefore, we skew load into a few active hot nodes to substantially boost the hot-node utilization and the overall energy efficiency.

The master node keeps track of metadata including the location of required data tables for queries, memory and disk usages of hot nodes. The master node is responsible for monitoring data access patterns, from which popular tables are discovered. The access patterns enable the prefetching module (see Section 4.1.4) to make the most energy-efficient prefetching decisions.

GreenDB consists of four software modules, namely, the query manager (see Section 4.1.3), the data prefecthing module (see Section 4.1.4), the energy-cost model (see Section 4.1.5), table replication module, and power manager (see Section 4.1.6).

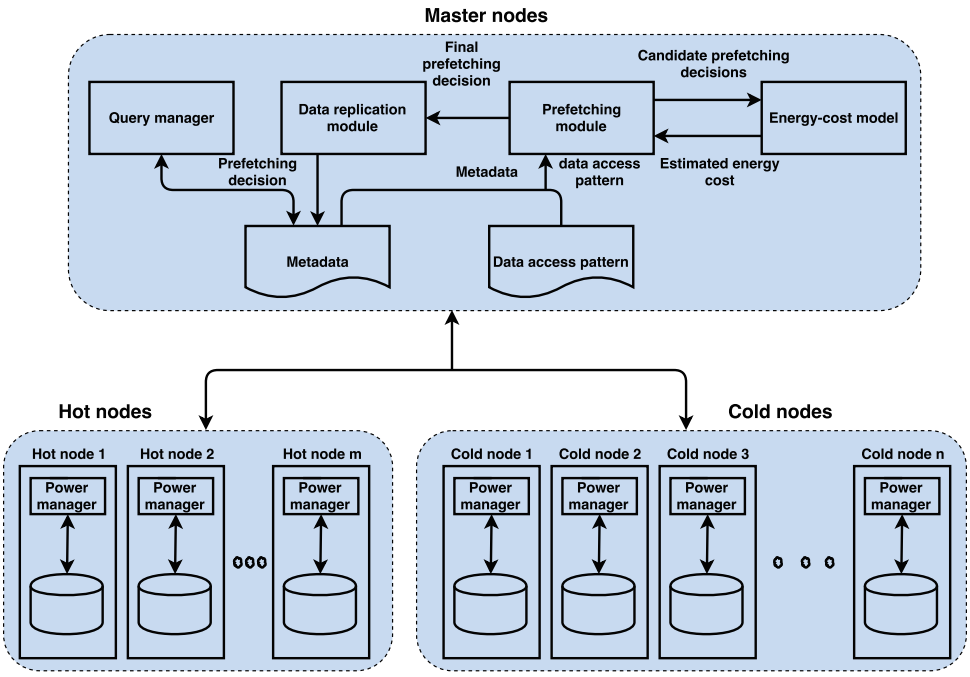


Figure 4.1: The architecture of the GreenDB system.

Queries issued by database applications are managed by the query manager. The data prefecthing module is focused on fetching popular data tables from cold nodes into hot nodes. The energy-cost model offers an energy-saving guideline for the prefetching module. The table replication module creates duplicated tables on hot nodes according to prefetching decisions. The power manager in each database nodes is responsible to decide the power state of each node.

To improve energy efficiency, the prefetching module makes use of an energy cost model to estimate energy overhead and impact of prefectching decisions. After a decision is made by the prefetching module, the data replication module duplicates popular tables from cold into hot nodes in a batch manner.

The functionality of the table replication module is straightforward - creating replicas of popular tables from the cold into hot nodes according to prefetching decisions made by the prefetching module. The duplicated tables enable hot nodes to serve as caching nodes of the cold ones in database clusters.

4.1.3 Query Manager

The query manager handles queries issued by database clients. Sophisticated queries may be partitioned by the query manager to form multiple sub-queries. The query manager relies on the metadata manager to determine the location of data to be accessed by queries. If database tables are residing in a hot node, the queries can be immediately answered. Otherwise, the corresponding cold node should be waked up to fetch and duplicate the tables to a hot node.

The responsibility of the query manager is three-fold. First, it decomposes complicated queries issued by database clients into small pieces referred to as subquery. Second, the query manager coordinates with the metadata manager to locate data from database nodes for each subquery.

To keep cold nodes stay in the low-power mode in a long time period, GreenDB aims to mathematically and accurately discover popular data to be duplicated from cold nodes to hot nodes. We design a model, which estimates the cost of fetching requested data from multiple cold nodes into hot nodes. When free space of the hot nodes is sufficient, popular data are replicated to the hot nodes in a greedy manner. In such an initial scenario where space is abundant, there is no need for GreenDB to remove any data from the hot nodes. When the hot node space becomes insufficient, GreenDB kicks in a data replacement strategy so that data with low access rate are replaced by new popular data on the hot nodes. For example, when a selection query is decomposed into small sub-queries involved with a few data tables, the GreenDB makes an effort to fetch the accessed data tables into the hot nodes if the tables are residing in the cold nodes. If subsequent queries are accessing these data tables, the queries can be served by the hot nodes rather than the cold nodes that have been transitioned into the standby mode.

4.1.4 Prefetching Module

The prefetching module periodically coordinates with the meatdata manager and the access pattern manager to make prefectching decisions for database clusters.

Table 4.1 Notation for the Description of the Prefetching Module.

Notation	Desription
L	Current lookahead. $q \in L$ is a reference in the lookahead
table(q)	Talbe accessed in reference $q \in L$
cold node(q)	Cold node in which *table*(q) is residing
Q	Subset of the lookahead Q; for any q in Q, table(q) is active, i.e., $\forall$ q $\in Q$: table(q) is active
H	A set of tables present in the hot node
$E_{saving}(\tau)$	Energy saving contributed by prefetching table τ
Q^+	For any q in Q^+, we have node(q) $\in Q$, $E_{saving}(\tau) > 0$,q$\notin H$, and$\exists\ q \in Q$: *table(q)* = τ
H^+	The set of tables with the highest energy savings in $Q^+ \cup H$

The prefetching module has two salient features. First, the module keeps track of the access patterns of records in tables to pinpoint popular data. Second, the module creates replicated copies of popular tables to be placed on the most appropriate hot nodes. The decisions of making replicated tables largely depends on potential energy savings, which are predicted by the energy-cost model (see Section 4.1.5).

This module analyzes the four important factors to make decisions on fetching data tables from cold nodes. The factors include the locations of candidate tables, the popularity of data accessed by queries, estimated energy savings, and query predictions. Note that energy savings of fetching tables are estimated by the energy-cost model (see Section 4.1.5). Query predictions are made possible by examining historical query access patterns. Query-prediction algorithms can be readily plugged into GreenDB (see Fig. 4.1) to governs the process of prefetching tables. GreenDB takes a greedy approach. Thus, when a query accesses a table, the prefetching module aggressively fetches all tables relevant to the accessed table from the cold nodes.

The existing prefetching scheme in *PostgreSQL* relies on the *synchronous replication* module to maintain duplicated data between cold and hot nodes. Such a synchronous-based prefetching strategy (or *SyncPrefetch* for short) in PostgreSQL is expensive when it comes to maintaining replicated data on hot nodes. Before committing an update transaction, *SyncPrefetch* must obtain exclusive locks on all the copies. The transaction may have to send lock requests to cold nodes, waiting for the locks to be granted. During such a potentially long time period,

SyncPrefetch continues holding all the other locks. If the cold nodes or communication links fail, the transaction cannot commit until all the nodes storing modified data are recovered the failure. Even if the locks are obtained readily without encountering any failure, committing a transaction has to deliver a few additional messages as specified by the commit protocol.

Unlike *SyncPrefetch*, the prefetching scheme in GreenDB takes the full advantage of asynchronous replications to cut energy cost of fetching hot data from cold nodes. A similar scheme has been applied in database systems [47]. Updated data are initially kept in hot nodes to avoid excessive power-state transitions in cold nodes. When new popular data are about to be transferred from the cold nodes to the hot ones, the update data will be synchronized back to the cold nodes. GreenDB's synchronize replicated data between hot and cold nodes in a lazy manner, thereby improving system energy efficiency. GreenDB offers energy savings, because the prefetching module in GreenDB not only allows the cold nodes to stay in the low-power mode for an increased time period, but also substantially reduces the number of power-state transitions in the cold nodes.

Now we propose our prefetching algorithm. The primary function is to prefetch the most energy-saving tables into the hot node.

Algorithm 3 GreenDB Prefetching: $Prefetching()$

Require:
a query q
GreenDB with m **cold nodes**
1: **if** $table(q)$ is present in hot node **then**
2: update L;
3: query q is serviced by hot node with $table(q)$;
4: **else**
5: **if** $cold\ node(q)$ is in standby state **then** power up $cold\ node(q)$;
6: **end if**
7: Fetch $table(q)$ to serve query q;
8: **end if**
9: Estimate the energy savings of queries $Q \in L$;
10: Update the energy savings of tables in the hot node;
11: Fetch tables in $Q^+ \cap H^+$
12: **if** disks of the hot node is full **then** Evicting the blocks in $H - H^+$ with the lowest energy savings as necessary;
13: **end if**

The prefetching algorithm (see Algorithm 3) outlined above initializes the query and the number of cold nodes to q and m, respectively . Next, if the $table(q)$ is resident in the hot node, it server the query by hot node, otherwise it will power up the cold node in which the $table(q)$ is

located and fetch the $table(q)$ into the hot node from the cold node(see Lines 1-7). Finally, we estimate the energy-efficiency of the tables in the lookahead table, fetch most energy-efficiency tables of whole GreenDB into the hot node if they are not kept in the hot node (see Lines 9-11). Unpopular tables are evicted from the hot nodes to release space for newly fetched ones (see Line 12).

The above prefetching algorithm will not be triggered, if prefetching tables has detrimental impacts on the energy efficiency of database clusters.

4.1.5 Energy-cost Model

To facilitate the replacement strategy in GreenDB, we delve into an energy-cost model projecting energy savings offered by keeping candidate data tables into the hot nodes while shutting down the cold nodes. The energy-cost model provides guideline for GreenDB to (1) evict unpopular data tables from the hot nodes and (2) fetch popular tables from the cold nodes to improve energy efficiency.

The energy-cost model integrates profiling results and the prefetching algorithm to offer us an interface to estimate energy savings provided by data tables in hot nodes. Profiling results take into account of three factors - energy consumed on fetching data from cold nodes, energy spent by hot nodes to store prefetched data tables, and energy overhead caused by power-state transitions. Our energy-cost model leverages the profiling data of energy consumption to estimate energy savings of offered by prefetched table.

To build the energy-cost model through an energy profiling study, we independently connect hot nodes, cold nodes, and network interconnection to three power sockets; the displays are plunged into a separate power socket, ensuring that measured energy readings is contributed by individual devices being gauged. The power meter employed in this study is TS-836A Plug Energy Watt Voltage Amps Meter (see TABLE II for details).

4.1.6 Power Manager

The power manager in GreenDB dynamically and automatically changes the power mode of cold nodes to converse energy. For instance, if a cold node has been sitting idle for a given

threshold, the power manager immediately transition the node into the low-power mode. Such a threshold can be automatically adjusted in GreenDB depending on workload conditions. For another example, prior to prefetching tables from a cold node to a hot node, the power manager wakes up the cold node.

A power-consumption monitor is built-in each of cold nodes to keep track of access activities in the cold nodes. The monitor initiates power-transition requests to the corresponding power manager of a cold node. Upon the arrivals of power-down requests, the manager immediately powers down the cold nodes. Since each cold node has its individual access history, the power manager dynamically adjusts spin-down intervals according to the access activities, thereby improving the energy efficiency of each cold node.

Table 4.2 Power Meter Specifications

	TS-836A Power Meter Specifications
Measurement of consumption	$0.00 \sim 9999.99$ KWh
Voltage display range	$0V \sim 9999$V
Current range	$0.000A \sim 15.000$A
Frequency display	$0Hz \sim 9999$Hz
Wattage display (Watts)	$0 \sim 1800$W

4.2 Energy-Efficient Prefetching

We develop in *GreenDB* an energy-saving prediction model, which governs the energy-saving calculation in *Steps* 9 and 10 in algorithm $Prefetching$. The prediction model along with the calculation module is indispensable for GreenDB's prefetching module, because the model speculates the amount of energy conserved by fetching candidate tables from cold nodes into hot nodes. The model also calculates potential cost of caching a hot-node-resident table rather than evicting the table from its hot node. Table 4.3 summarizes the notation used throughout this section.

To analyze circumstances under which prefetching database tables can yield energy savings, we focus on a single referenced table stored in a node. Let $Q_j \subseteq L$ be a set of queries accessing tables in the jth node. Thus, Q_j - a subset of lookahead L - can be defined as

Table 4.3 Notation for the Description of the Energy-Saving Calculation Module.

Notation	Desription
Q_j	A set of references accessing tables in the *j*th cold node
$Q_{k,j} \subseteq L$	A set of references accessing the *k*th table $\tau_{k,j}$ in the *j*th cold node
$\tau_{k,j}$	The *k*th table in the *j*th cold node
T_{BE}	Break-even time. Minimum idle time required to compensate the cost of entering the standby state
T_{ij}	Active time period serving the *i*th request issued to the *j*th cold node
t_{ij}	Time spent serving the *i*th request issued to the *j*th cold node
α_{ij}	Time spent in the idle period prior to the *i*th request accessing a table in node *j*
I_{ij}	An idle period prior to the *i*th request accessing a table in the *j*th cold node
n_j	The total number of requests (in the lookahead) issued to the *j*th cold node
Ωj	A set of cold node access activities for references in Q_j
$time(\tau_{kj})$	Active time period to serve a request accessing table τ_{kj}
$table(\tau_j)$	A table accessed during the active period T_j
T_D	Time to transition from active/idle to standby
T_U	Time to transition from standy to active mode
E_D	Energy overhead of transitioning from active/idle to standby
E_U	Energy overhead of transitioning from standby to active mode
P_A, P_I, P_S	Node power in the active, idle, and standby mode

$$Q_j = \{q \,|(q \in L) \cap (node(q) = j) \cap (\mathrm{table}(q) \notin H)\},$$

where $node(q) = j$ means query q is accessing the jth node; $\mathrm{table(q)} \notin \mathrm{H}$ indicates that the table accessed by query q has not been fetched into hot node denoted as H.

Given a set $Q_{kj} \subseteq Q_j$ of queries accessing the kth table τ_{kj} in the jth cold node, we derive the energy saving $E_{Saving}(\tau_{ij})$ achieved by fetching table τ_{kj} from the cold node into a hot node. Q_{kj} is comprised of all the queries referencing a common table τ_{kj} that is not present in the hot node; therefore, Q_{kj} can be formally expressed as

$$Q_{k,j} = \{q \,|(q \in Q_j) \cap (table(q) = \tau_{k,j}) \cap (\tau_{k,j} \notin H)\},$$

where $table(q) = \tau_{k,j}$ means query q is accessing table $\tau_{k,j}$; $\tau_{k,j} \notin H$ suggests that table $\tau_{k,j}$

is not residing in the hot node. Intuitively, energy savings $E_{saving}(\tau_{kj})$ can be computed by considering the energy consumption incurred by each query in Q_{kj}.

4.2.1 Two Energy Saving Principles

We introduce two energy saving principles inspiring the development of GreenDB.

Energy-Saving Principle 1. GreenDB aims not only to enlarge idle periods but also to increase the number of idle periods in database nodes. Idle periods should be larger than break-even time T_{BE} to offset the cost of entering the low-power state. Principle 1 can be realized by combining two adjacent idle periods to form a single idle period being larger than T_{BE}. GreenDB prefetches a table accessed by adjacent queries separated by idle periods, thereby forming a large idle time allowing the node to enter the low-power state to conserve energy.

Energy-Saving Principle 2. GreenDB strives to reduce the number of power-state transitions. The energy efficiency of database nodes can be improved by minimizing the energy cost of powering up/down nodes. GreenDB reduces the number of node power transitions while enlarging node idle times. We implement the second principle in GreenDB by combining two adjacent standby periods to eliminate unnecessary power-state transitions between the two adjacent standby periods.

4.2.2 Modeling Energy Savings

Given query list Q_j and table τ_{kj}, we investigate four cases where a query in Q_j contributes to positive energy savings by the virtue of prefetching table τ_{kj}. The four cases exploit the above two energy saving principles to conserve energy in nodes.

Let $\Omega_j = \{\alpha_{1j}, t_{1j}, \alpha_{2j}, t_{2j}, \ldots \alpha_{ij}, t_{ij}, \ldots \alpha_{nj,j}, t_{nj,j}\}$ be a list of idle and active time intervals for queries in Q_j, where for t_{ij} is the active time spent in serving the ith query issued to cold node j, α_{ij} is an idle period prior to the ith query accessing a table in the jth node, and n_j is the total number of queries issued to node j. We denote τ_{ij} as a table accessed during active period t_{ij}.

After table τ_{kj} is fetched to the hot node, cold node j's set

$\Omega_j = \{\alpha_{1j}, t_{1j}, \alpha_{2j}, t_{2j}, ...\alpha_{ij}, t_{ij}, ...\alpha_{nj,j}, t_{nj,j}\}$

of access activities for queries in Q_j must be updated by deleting any interval $t_{ij} \in \Omega_j$ in which τ_{kj} is accessed.

The energy-saving model makes use of the following definitions:

- P_A, P_I, and P_S represent the node power consumption in the active, idle, and standby (i.e., low-power) modes. Let T_D and T_U be time spend in transitioning to the standby and active modes; let E_D and E_U be energy overhead to transition to standby and active.
- $E_{non\text{-}PM}$ denotes energy consumption of anwering queries without deploying GreenDB.
- In case prefetching τ_{ij}, E_{Cold} denotes energy consumption of the jth node in periods t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$.
- E_{Hot} represents energy consumption of a hot node accessing prefetched τ_{ij}.
- $time(\tau_{k,j})$ is the amount of active time spent answering a query accessing table τ_{ij}.

Energy savings, $E_{Saving}(\tau_{ij})$, contributed by prefetching τ_{ij} can be written as:

$$E_{Saving}(\tau_{ij}) = E_{non\text{-}PM} - (E_{Cold} + E_{Hot}) \ , \tag{4.1}$$

where energy saving $E_{Saving}(\tau_{ij})$ is the difference between energy consumption $E_{non\text{-}PM}$ and energy cost E_{Hot} coupled with E_{Cold}.

Let us build the energy saving model with respect to the following five cases. The first three cases demonstrate scenarios of *energy saving principle 1*, under which long idle periods are created (i.e., longer than T_{BE}) by prefetching τ_{ij} to combine the ith and $(i+1)$th idle periods. We pay attention to the ith active period T_{ij} and two periods α_{ij} and $\alpha_{(i+1)j}$ (i.e., the ones adjacent to T_{ij}). Cases $1-3$ share two common conditions: (a) both α_{ij} and $\alpha_{(i+1)j}$ are larger than zero and (b) the summation of t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ is larger than break-even time T_{BE}.

Case 1: Both the ith and $(i+1)$th idle periods are equal to or smaller than break-even time T_{BE}. Thus, we have $0 < \alpha_{ij} \leq T_{BE}$, $0 < \alpha_{(i+1)j} \leq T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$. This condition implies that the jth cold node is in the idle mode during α_{ij} and $\alpha_{(i+1)j}$. Energy

consumption experienced by the node in active period t_{ij} is $P_A \cdot t_{ij}$. Hence, energy consumption $E_{non\text{-}PM}$ in case 1 can be expressed as:

$$E_{non\text{-}PM} = P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) + P_A \times t_{ij}. \tag{4.2}$$

When table τ_{ij} is prefetched, a large (i.e., larger than T_{BE}) idle period can be formed by combining periods t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$. Therefore, E_{Cold} can be computed as the energy consumption of the jth cold node in the standby mode during t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$. Taking into account energy overhead of power-state transitions, we can calculate energy consumption E_{Cold} of cold node using the equation below:

$$E_{Cold} = P_S \times (\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} - T_D - T_U) + E_D + E_U, \tag{4.3}$$

where E_D and E_U are energy cost to power down and up the cold node; T_D and T_U are time spent in power transitions.

We assume that the hot and cold nodes are homogeneous; therefore, energy consumption E_{Hot} of the hot node accessing prefetched table τ_{ij} is

$$E_{Hot} = P_A \times t_{ij}. \tag{4.4}$$

$E_{Saving}(\tau_{ij})$ in case 1 can be determined by substituting (4.2)-(4.4) into (4.1). Hence, we have:

$$\begin{aligned} E_{Saving}(\tau_{ij}) &= P_I \times (\alpha_{ij} + \alpha_{(i+1)j}) \\ &\quad -P_S \times (\alpha_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U. \end{aligned} \tag{4.5}$$

Case 2: The ith idle period is equal to or smaller than break-even time T_{BE}; the $(i+1)$th idle period is larger than T_{BE}. Formally, we have $0 < \alpha_{ij} \leq T_{BE}$,$\alpha_{(i+1)j} > T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$.

The jth cold node in this case is transitioned into standby during $\alpha_{(i+1)j}$, since $\alpha_{(i+1)j}$ is larger than T_{BE}. The energy consumption of the cold node in $\alpha_{(i+1)j}$ is expressed as (see the

third term on the right hand side of (4.6) below). Thus, the energy consumption $E_{non\text{-}PM}$ of cold node during t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ is

$$E_{non\text{-}PM} = P_I \times \alpha_{ij} + P_A \times t_{ij} + (P_S \times (\alpha_{(i+1)j} - T_D - T_U) + E_D + E_U) \tag{4.6}$$

We derive $E_{Saving}(\tau_{ij})$ in case 2 by substituting (4.6), (4.3), and (4.4) for for $E_{non\text{-}PM}$, E_{Trans}, and $E_{GreenDB}$. Thus, we have

$$E_{saving}(\tau_{ij}) = P_I \times \alpha_{ij} - P_S \times (\alpha_{ij} + \tau_{ij}) \tag{4.7}$$

Case 3: The ith idle period is larger than T_{BE}; the $(i+1)$th idle period is equal to or smaller than T_{BE}. Case 3's conditions can be formally expressed as: $\alpha_{ij} > T_{BE}$,$0 < \alpha_{(i+1)j} \leq T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$.

The energy saving $E_{Saving}(\tau_{ij})$ in case 3 is very similar to that in case 2 except that the jth cold node is transitioned into standby during α_{ij} rather than $\alpha_{(i+1)j}$. Consequently, energy saving $E_{Saving}(\tau_{ij})$ can be written as

$$E_{saving}(\tau_{ij}) = P_I \times \alpha_{(i+1)j} - P_S \times (\alpha_{(i+1)j} + T_{ij}) \tag{4.8}$$

Case 4: This case shows a scenario where *energy saving principle 2* is applied to reduce power-state transitions by prefetching τ_{ij} to combine two adjacent standby periods α_{ij} and $\alpha_{(i+1)j}$.

In this case, both α_{ij} and $\alpha_{(i+1)j}$ are larger than T_{BE}, meaning that the jth cold node can be transitioned into standby during these two time intervals to conserve energy. Formally, we have $\alpha_{ij} > T_{BE}$,$\alpha_{(i+1)j} > T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$. Thus, energy consumption $E_{non\text{-}PM}$ of the jth cold node without deploying a hot node is

$$\begin{aligned} E_{non\text{-}PM} = {} & P_A \times t_{ij} + (P_S \cdot (\alpha_{ij} - T_D - T_U) + E_D + E_U) \\ & + \left(P_S \times (\alpha_{(i+1)j} - T_D - T_U) + E_D + E_U\right), \end{aligned} \tag{4.9}$$

where the second and third term on the right hand side of (4.9) are the energy consumed by the cold node in standby periods α_{ij} and $\alpha_{(i+1)j}$, respectively. With a hot node in place, energy consumption E_{Cold} and E_{Hot} in this case are the same as in case 1 (see also (4.3) and (4.4)). Therefore, the energy saving $E_{Saving}(\tau_{ij})$, is derived from $E_{non\text{-}PM}$ (see (4.9)), E_{Cold}, and E_{Hot} as

$$E_{Saving}(\tau_{ij}) = E_D + E_U - P_S \times (T_D + T_U + t_{ij}). \tag{4.10}$$

Case 5: This case summarizes scenarios where prefetching a table may impose negative impacts on energy efficiency. If the summation of t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ is smaller than or equal to T_{BE} (i.e., $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} \leq T_{BE}$), then prefetching table τ_{kj} may reduce energy efficiency.

Because $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} \leq T_{BE}$, cold node j stays in the idle mode for the periods of t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$. If table τ_{kj} is prefetched to the hot node, energy consumption E_{Cold} of cold node j in the three periods is

$$E_{Cold} = P_I \times (\alpha_{ij} + t_{ij} + \alpha_{(i+1)j}). \tag{4.11}$$

The values of $E_{non\text{-}PM}$ and E_{Hot} are the same as those of case 1 (see also (4.2)). Applying $E_{non\text{-}PM}$, E_{Trans} and $E_{GreenDB}$ to (4.1), we estimate the negative energy-saving impact as:

$$E_{saving}(\tau_{ij}) = -P_I \times t_{ij} \tag{4.12}$$

Given the above cases, set Ω_{kj} of cold nodes activities for queries accessing table τ_{kj} in cold node j can be partitioned into the following four disjoint subsets,

$$\Omega_{k,j} = \Omega_{k,j,1} \cup \Omega_{k,j,2} \cup \Omega_{k,j,3} \cup \Omega_{k,j,4} \cup \Omega_{k,j,5} \tag{4.13}$$

where $\Omega_{k,j,1}$, $\Omega_{k,j,2}$, $\Omega_{k,j,3}$, $\Omega_{k,j,4}$ and $\Omega_{k,j,5}$ contain active time periods that respectively satisfy the conditions of the five energy-saving cases. The five subsets are expressed as:

1. for case 1:

$$\begin{aligned}\Omega_{k,j,1} &= T_{ij}|(\tau_{ij}) = \tau_{k,j} \cap 0 < \alpha_{ij} \leq T_{BE} \cap 0 < \alpha_{(i+1)j} \\ &\leq T_{BE} \cap \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

2. for case 2:

$$\begin{aligned}\Omega_{k,j,2} &= T_{ij}|(\tau_{ij}) = \tau_{k,j} \cap 0 < \alpha_{ij} \leq T_{BE} \cap \alpha_{(i+1)j} \\ &> T_{BE} \cap \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

3. for case 3:

$$\begin{aligned}\Omega_{k,j,3} &= T_{ij}|(\tau_{ij}) = \tau_{k,j} \cap \alpha_{ij} > T_{BE} \cap 0 < \alpha_{(i+1)j} \\ &\leq T_{BE} \cap \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

4. for case 4:

$$\begin{aligned}\Omega_{k,j,4} &= T_{ij}|(\tau_{ij}) = \tau_{k,j} \cap \alpha_{ij} > T_{BE} \cap \alpha_{(i+1)j} \\ &> T_{BE} \cap \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

5. for case 5:

$$\begin{aligned}\Omega_{k,j,4} &= T_{ij}|(\tau_{ij}) = \tau_{k,j} \cap \alpha_{ij} > T_{BE} \cap \alpha_{(i+1)j} \\ &> T_{BE} \cap \alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}\end{aligned}$$

Now we derive energy saving $E_{Saving}(\tau_{kj})$ offered by fetching table τ_{kj} from cold node j to the hot node. Thus, we obtain $E_{Saving}(\tau_{kj})$ from (4.5), (4.7), (4.8), (4.10), (4.11) and (4.12), where the last item on the right hand side is the energy overhead of fetching τ_{kj} from cold node j to the hot node.

$$
\begin{aligned}
&E_{Saving}(\tau_{k,j}) = \sum_{T_{ij} \in \Omega_{k,j}} (E_{Saving}(\tau_{ij})) \\
&= \sum_{T_{ij} \in \Omega_{k,j,1}} (P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) \\
&\quad -P_S \cdot (\Omega_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U) \\
&\quad + \sum_{T_{ij} \in \Omega_{k,j,2}} (P_I \cdot \alpha_{ij} - P_S \cdot (\alpha_{ij} + t_{ij})) \\
&\quad + \sum_{T_{ij} \in \Omega_{k,j,3}} \big(P_I \cdot \alpha_{(i+1)j} - P_S \cdot (\alpha_{(i+1)j} + t_{ij})\big) \\
&\quad + \sum_{T_{ij} \in \Omega_{k,j,4}} (E_D + E_U - P_S \cdot (T_D + T_U + t_{ij}).) \\
&\quad -P_I \cdot \sum_{T_{ij} \in \Omega_{k,j,5}} t_{ij} - P_A \cdot \text{time}(\tau_{k,j})
\end{aligned}
\tag{4.14}
$$

4.2.3 Calculating Energy Savings

We show how to plugin the model (see Section 4.2.2) into an algorithm to calculate energy conserved by *GreenDB*. Given the kth table τ_{kj} residing in cold node j, Algorithm 4 computes energy savings of prefetching table τ_{kj} from the cold nodes to the hot node.

All of the energy-saving cases presented in Section 4.2.2 are explicitly handled from Steps 1 through 14 (See Algorithm in 4: Case 1 in Line 7, Case 2 in Line 9, Case 3 in Line 12, Case 4 in Line 15), whereas Step 18 addresses the issue of a negative impact on energy savings.

The time complexity of computing energy savings is $O(m \times n_j)$, where m is the number of cold node and n_j is the number of queries in the look-ahead corresponding to the j cold node.

4.3 Experimental Results

To quantitatively evaluate *GreenDB*, we first compare our prefetching strategy with the one implemented in Postgresql using the build-in stream replication scheme (see Section 4.3.2). We measure energy consumed by replicating one data table from a cold node to a hot node. The purpose of this set of experiments is to investigate an energy-efficient way of managing, replicating, and storing data in GreenDB. Creating data replications on hot nodes has a strong

Algorithm 4 Calculating Energy Savings

Require:
table $\tau_{k,j}$
cold node j
a set Ω_j **of queries accessing cold** $node_j$
Ensure:
$E_{Saving}(\tau_{k,j})$

update $E_{Saving}(\tau_{k,j}) \leftarrow 0$;
for $j = 1$ to m **do**
  for $i = 1$ to n_j **do**
    if $alpha_{ij} + t_{i,j} + i_{(i+1)j} > T_{BE}$ **then**
      if $0 < i_{i,j} \leq T_{BE}$ **then**
        if $0 < i_{i,j} \leq T_{BE}$ **then**
          if $0 < i_{(i+1)j} \leq T_{BE}$ **then**
            $E_{saving}(\tau_{ij}) = P_I \times (\alpha_{ij} + \alpha_{(i+1)j})$
            $-P_S \cdot (\alpha_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U$
          else
            $E_{saving}(\tau_{ij}) = P_I \cdot \alpha_{ij} - P_S \cdot (\alpha_{ij} + \tau_{ij})$
          end if
        else
          $E_{saving}(\tau_{ij}) = P_I \cdot \alpha_{(i+1)j} - P_S \cdot (\alpha_{(i+1)j} + T_{ij})$
        end if
      else
        $E_{saving}(\tau_{ij}) = E_D + E_U - P_S \times (T_D + T_U + t_{ij})$
      end if
    else
      $E_{saving}(\tau_{ij}) = -P_I \times t_{ij}$
    end if
  end for
end for
return $E_{saving}(\tau_{ij}) - P_A \times time(\tau_{kj})$

impact on energy consumption and; therefore, we shed light on the impacts of vital factors on the two evaluated prefetching strategies in GreenDB. The factors considered in our experiments include data size (see Section 4.3.3), the number of nodes (see Section 4.3.4), hit rate (see Section 4.3.5), inter-arrival times (see Section 4.3.6), and the number of disks in a cold node (see Section 4.3.7).

4.3.1 Experimental Setups

All the storage nodes managed by GreenDB are running the PostgreSQL database system within CentOS. PostgresSQL, a highly extensible object-relational database management system, securely stores and retrieves a variety of data. Prostgres is capable of processing workloads of small-scale applications as well as large Web-based applications. We maintain a dedicated computing environment to test our prefetching strategy implemented in GreenDB - a database system running on a cluster. The focus of our empirical study is to explore and validate the energy efficiency of GreenDB on clusters. PostgresSQL facilitates a convenient way of managing and transferring replicated data on hot nodes in GreenDB. We compare our energy-efficient prefetching scheme with the existing one offered in PostgresSQL. Table 4.4 shows the specifications of the cluster running the tested database system.

Table 4.4 Testbed Configurations.

	OptiPlex 3020 MT/SFF Technical Specifications
CPU	Intel 4th Core i5-4570 Quad Core@3.20GHz
Memory	4GB Non-ECC 1600MHz DDR3 SDRAM
Hard Drives	Seagate KC47-500GB SATA (7.200 RPM)
Operating System	CentOS 6.5(Final) Linux kernel $2.6.32 - 431.el6.x86_64$
Database System	PostgreSQL 9.3.5

4.3.2 Prefetching Data to Hot Nodes

The prefetching mechanism plays an important role in reducing energy consumption of database systems running on clusters. In this part of the study, we pay attention to the energy efficiency of the prefetching scheme that creates and transfers replicated data from cold nodes to hot ones

in GreenDB. In particular, we evaluate the impact of the data table size on energy savings offered by our prefetching strategy.

Recall that *SyncPrefetch* in PostgreSQL is expensive in terms of energy consumption (see Section 4.1.4). In this group of experiments, we show that the prefetching scheme in GreenDB employs asynchronous replications to minimize energy cost of fetching hot data from cold nodes.

For simplicity, we keep one hot node in GreenDB. Nevertheless, our findings indicate that the energy-efficiency trend of multiple-hot-node cases is similar to that of the single-hot-node case. The experimental results for multiple-hot-node scenarios can be found in Section 4.3.4.

GreenDB vs. SyncPrefetch

We compare the prefetching strategy in GreenDB with the existing Postgresql strategy referred to as *SyncPrefetch*, thereby quantifying energy cost of fetching a data table from a cold node to a hot node. The energy consumption of prefetching a data table is comprised of three parts: (1) energy cost of fetching the table from cold nodes, (2) energy consumed by transferring the table through the network interconnection, and (3) energy consumption of storing the table into the hot node.

Fig. 4.2 shows the energy consumption of prefetching a data table in GreenDB and SyncPrefetch. The results depicted in Fig. 4.2 indicate that GreenDB is superior to SyncPrefetch in terms of energy efficiency. More specifically, GreenDB improves the energy efficiency of SyncPrefetch by a factor up to 60.9 with an average of 52.5. We also observe from Fig. 4.2 that the energy consumption of both prefetching strategies goes up as the fetched table size increases.

GreenDB substantially reduces energy consumption of prefetching data by replacing the synchronous replication module in SyncPrefetch with the asynchronous one (see the details in Section 4.1.4). SyncPrefetch offers slim opportunities for hardware resources (e.g., CPUs, hard drives) in cold nodes to transition into the low-power state, thereby leading to high energy consumption. In contrast, the prefetching strategy in our GreenDB fetches hot data in a batch manner, which keeps the hardware resources in the energy-saving mode for a long time period after the hot-data prefetching process is completed.

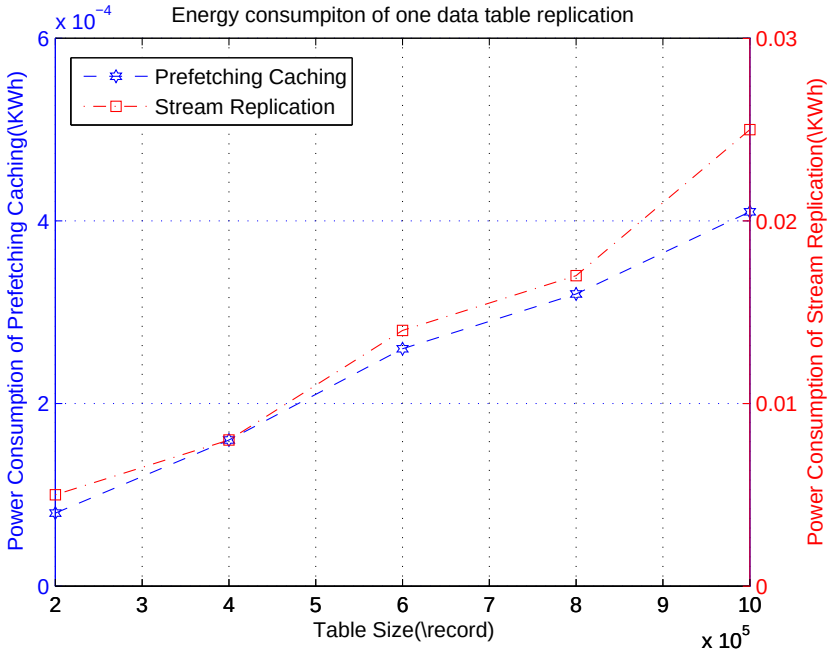


Figure 4.2: Energy consumpiton of Prefetching one data table from a cold node to a hot node.

Fig. 4.3 shows the energy-saving ratio of GreenDB over SyncPrefetch. The results plotted in Fig. 4.3 indicate that GreenDB conserves the energy consumption of SyncPrefetch by up to 98.4%. The energy-saving ratio is extremely high when the data table size is either small (e.g., $2 * 10^5$records) or large (e.g., $1 * 10^6$records). We conclude that GreenDB is more energy efficient than Postgresql's build-in SyncPrefetch.

Energy Consumption and Execution Time

Apart from energy efficiency, we investigate the performance of our GreenDB and SyncPrefetch. We measure the execution times of GreenDB and SyncPrefetch process a list of queries. For fair comparison, we ensure the identical list of queries are issued to the two schemes handling the same data tables.

Fig 4.4(c) reveals that our GreenDB is faster than SyncPrefetch during the data transfer from cold to hot nodes. For example, it takes less than 30 seconds to create a replicated table of 2×10^5 records on the hot node, whereas SyncPrefetch spends more than 300 seconds to accomplish the duplication process. When the table size increases to 10×10^5, SyncPrefetch's

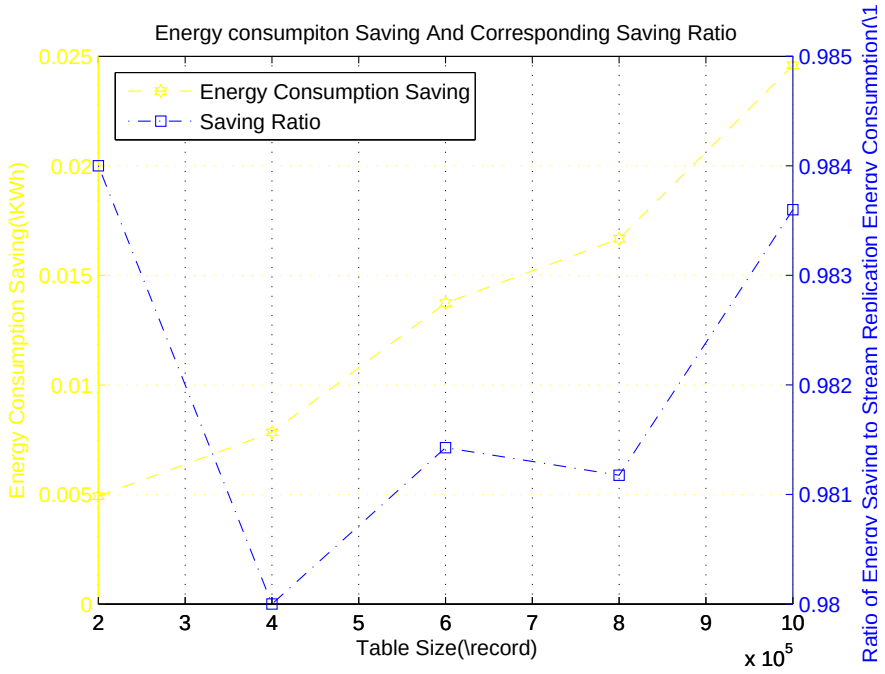


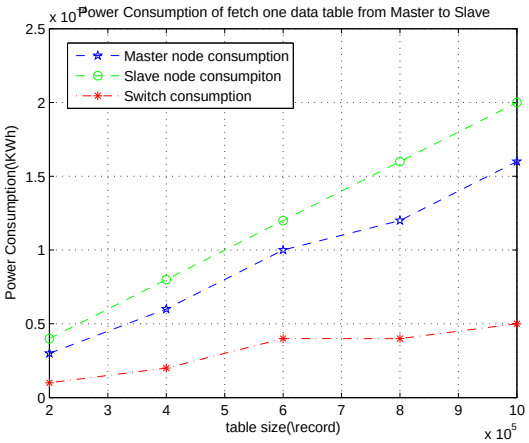
Figure 4.3: Energy consumpiton Saving And Corresponding Saving Ratio.

processing time is enlarged to 20 minutes, which is approximately 67 times longer than that of our GreenDB.

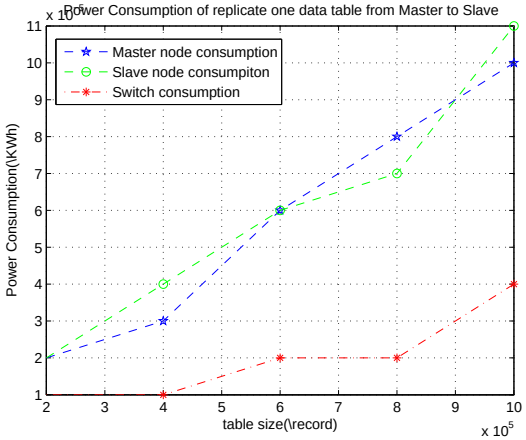
Energy consumption is a product of power and processing time, meaning that shortening the processing time is a practical way of boosting energy efficiency. Thus, reducing time spent in moving data tables from cold to hot nodes can minimize the energy consumption overhead of fetching hot data from cold nodes. Although Postgresql's SyncPrefetch maintain consistency between hot and cold nodes in a real-time manner, such consistency comes at the cost of high performance and energy overhead.

4.3.3 Data Table Size

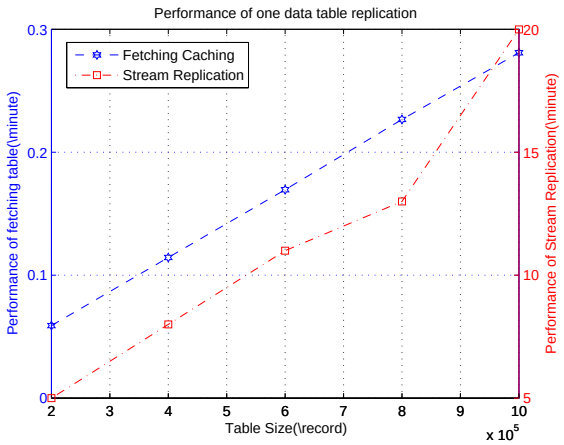
The second set of experiments is focused on evaluating the impacts of data table size (data size for short) on the energy savings offered by GreenDB after replicated tables are created on hot nodes. We set the number of cold nodes to four; we set the hit rate of the hot nodes to 25% (see Fig. 4.5(a)), 50% (see Fig. 4.5(b)), and 75% (see Fig. 4.5(c)), respectively. We compare our GreenDB with the two existing schemes - *PRE-BUD* [37] and *DPM* [28]. For comparison



(a) Power Consumption of fetch one data table from Cold node to Hot node



(b) Power Consumption of replicate one data table from Cold node to Hot node



(c) Comparison of performance for replicating one data table from Cold node to Hot node

Figure 4.4: Energy consumption and Performance profiling between fetching and replicating techniques

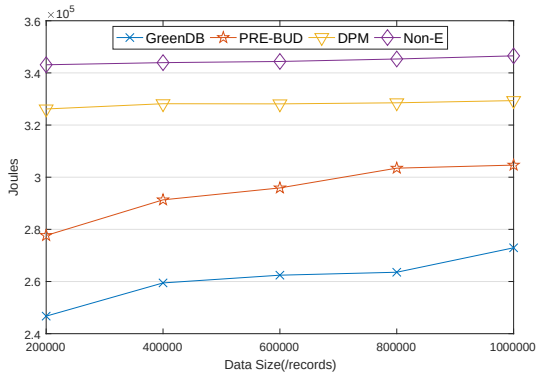
purpose, we also evaluate the database system where no energy-saving scheme is deployed; we refer to this case as *Non-Energy Saving* or *Non-E* for short.

Fig. 4.5 reveals that the data size imposes noticeable impact on the energy efficiency of GreenDB and PRE-BUD strategy when the hit rate is lower than 75%. Unlike GreenDB and PRE-BUD, the energy consumption of DPM and Non-E are insensitive to data size. The data table size has very little impact on energy efficiency on DPM and Non-E, because power management in DPM and Non-E has nothing to do with creating replicated data tables from cold to hot nodes.

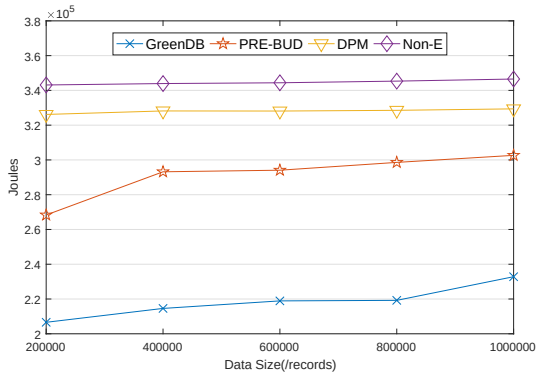
We observe from Fig. 4.5 that increasing the data size make the energy consumption go up. This trend is contributed by the fact that if incoming queries are unable to be served by a hot node, corresponding cold nodes will have to be waked up to moving data to the hot node. A large data size implies a long time spent moving the data table from the cold to hot node. It is worth noting that such a trend is diminishing when the hit rate is large (e.g., 75%), because the large hit rate cuts the chances of waking up cold nodes to move hot data.

Fig. 4.5 shows that regardless of the data size, a large hot-node hit rate (e.g., 75%) provides cold nodes with ample opportunity to stay in the low-power mode for a long period of time. This results suggest that a well-design prefetching algorithm play a key role in improving energy efficiency of database systems running on clusters, because the prefetching algorithm is responsible for improving the hit rate. Such a prefetching mechanism will be addressed in our future study.

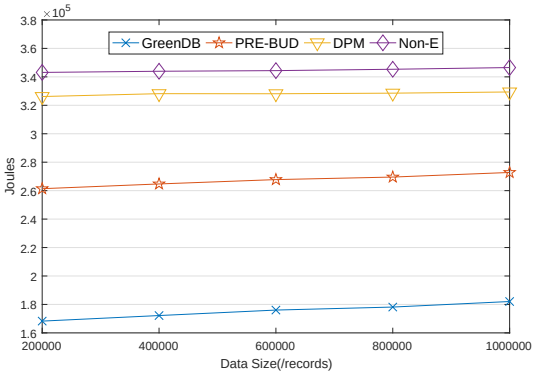
The results depicted in Fig. 4.5 indicates that our GreenDB outperforms PRE-BUD and DPM. And energy consumption of GreenDB increases slowly when hit rate reaches the level of 75%. GreenDB is superior to PRE-BUD, because PRE-BUD achieves high energy efficiency by aggressively turning off idle disks whereas GreenDB powered off the entire cold nodes rather than individual disks. In doing so, GreenDB not only conserves energy consumed by idle disks, but also saves energy caused by CPUs, memory, and other component in idle storage nodes. The experimental results confirm that small data table sizes coupled with high hit rates helps in improving energy efficiency of GreenDB.



(a) Total energy consumption of DBMS when the hit rate is 25%.



(b) Total energy consumption of DBMS when the hit rate is 50%.



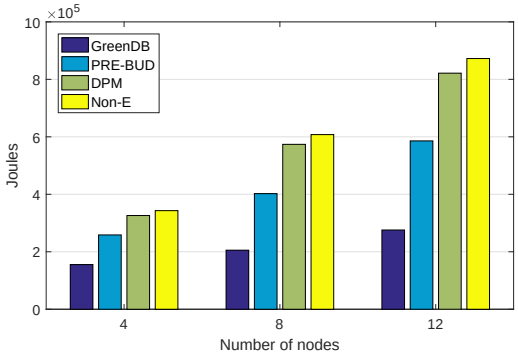
(c) Total energy consumption of DBMS when the hit rate is 75%.

Figure 4.5: Total energy consumption of database clusters while data size is varied for three different values of the hit rate: (a) 25%, (b) 50%, (c) 75%.

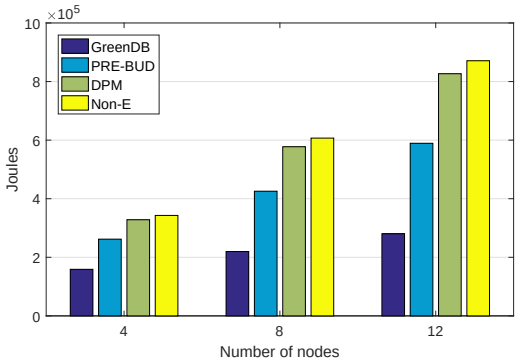
4.3.4 The Number of Nodes

Now we evaluate the impact of the number of nodes on GreenDB. The number of hot node is fixed at one; the number of cold nodes is set to 4, 8, and 12, respectively. The hit rate is

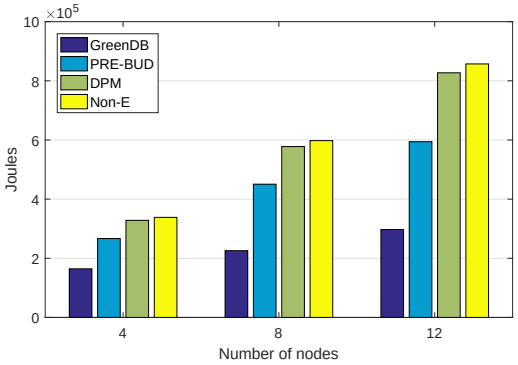
configured at the level of 85%. Fig. 4.6 shows the energy consumption of GreenDB, PRE-BUD, DPM, and Non-E under three data table sizes (i.e., 200K, 400K, and 800K records).



(a) Total energy consumption of database clusters while data size is fixed at 200K records



(b) Total energy consumption of database clusters while data size is fixed at 400K records



(c) Total energy consumption of database clusters while data size is fixed at 800K records

Figure 4.6: Total energy consumption of database clusters while the number of nodes is varied. Data size is fixed at (a) 200K, 400K, 800K records.

Not surprisingly, we discover from Fig. 4.6 that when the number of cold nodes goes up, energy savings offered by GreenDB becomes more pronounced. For example, let us consider a case where the data table size is 800K records (see Fig 4.6(c)). The GreenDB reduces the

energy consumption of Non-E by 51.4% and 65.3% when the number of the number of cold nodes is 4 and 12, respectively. This energy-efficiency trend is expected, because an increasing number of cold nodes are kept in the low-power mode to reduce energy consumption of the database system. After the hot node fetches popular tables from the cold nodes, lightly loaded cold data nodes are more likely to be switched into the power-saving mode to conserve energy by increase idle periods that are longer than the break even time.

Increasing the number of cold nodes to optimize energy efficiency of GreenDB comes is likely to have two side effects. First, GreenDB fetches a small amount of popular tables from each cold node to the hot node, which increasingly becomes a performance bottleneck. Second, when we expend the number of cold nodes, the storage capacity of the hot node may becomes insufficient for all the hot tables flooding from the cold nodes. When the performance issue occurs, we should increase the number of hot nodes in GreenDB to match the cold nodes' needs.

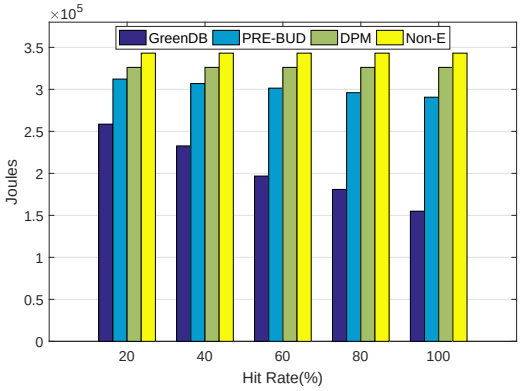
Comparing Figs. 4.6(a) and (c), we observe that GreenDB deliver higher energy efficiency when data table size is larger (e.g., 800K records). This trend is consistent with the one depicted in Fig. 4.5 (see Section 4.3.3).

4.3.5 Hit Rates

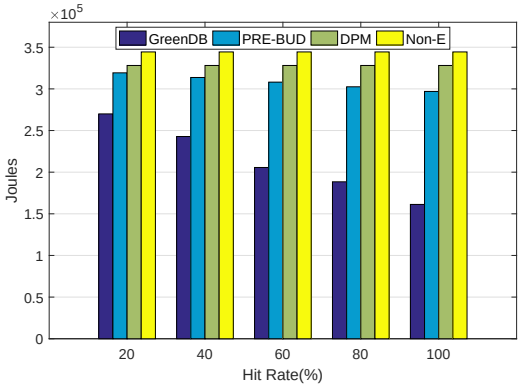
In this set of experiments, we choose to investigate the hit rate impact on the energy efficiency of the evaluated database system. Hit rate measures the probability that a query can be serviced by hot nodes rather than cold nodes. We configure the data table size at $200K$, $400K$ and $600K$ records. The number of cold nodes is set to four. Fig. 4.7 shows the total energy consumption as a function of hit rate values.

We observe from Fig. 4.7 that a high hit rate improves GreenDB's energy efficiency. For instance, when the hit rate is set to 80% and 20%, GreenDB reduces energy consumption of Non-E by 41.0% and 15.1%, respectively (see Fig. 4.7(c)). This trend is expected, because increasing hit rate enlarges the time period in which cold nodes are kept in the low-power state. A low hit rate leads to a strong likelihood that cold nodes are frequently waked up to serve requests. These experimental results are consistent with those plotted in Fig. 4.5.

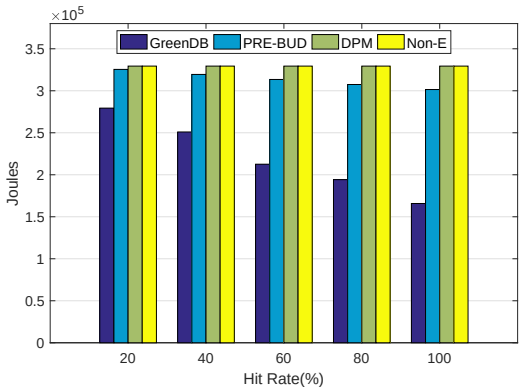
We argue that it is unrealistic to achieve a hit rate of 100% in dynamic workload conditions. Nevertheless, a 100% hit rate may be accomplished if query access patterns are given a priori. It is feasible to apply comprehensive prefetching algorithms to achieve hit rates as high as 80%.



(a) Total energy consumption when the data size is fixed at 200K records



(b) Total energy consumption when the data size is fixed at 400K records



(c) Total energy consumption when the data size is fixed at 600K records

Figure 4.7: Total energy consumption for different hit rate values where the data size is fixed at: (a) 200K records, (b) 400K records, (c) 600K records.

4.3.6 Inter-arrival Delays

Now we study the impact that query inter-arrival rates on the energy savings of GreenDB. Fig. 4.8 shows the total energy consumption of the database system as a function of the inter-arrival delay. Again, we set the number of cold nodes as four; the data size is fixed at $1,000,000$ records. The hit rate is set to 55%, 65%, 75%, and 85%, respectively.

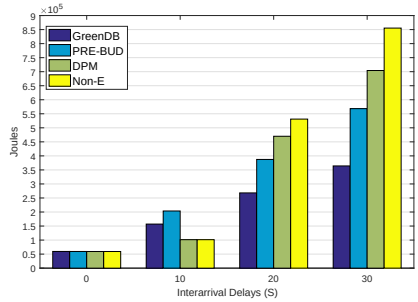
Fig. 4.8 demonstrated that when the query inter-arrival delay is less than 10 seconds, GreenDB and PRE-BUD are unable to yield any energy savings. Such a low energy efficiency is attributed by the lack of large idle windows that keep cold nodes in the energy-saving state. Unfortunately, in these scenarios, GreenDB and PRE-BUD end up consuming more energy than the traditional DPM due to the prefetching overhead.

Both GreenDB and PRE-BUD start delivering energy savings when the inter-arrival delay increase. For example, Fig. 4.8(d) confirms that when the inter-arrival delay is as large as 30 seconds, GreenDB and PRE-BUD conserve energy by 57.4%, 33.6%, respectively; DPM saves energy by a meager percentage of 17.7%. Intuitively, DPM relies on large idle times between consecutive queries to achieve energy savings; GreenDB, on the other hand, proactively creates large idle windows for cold nodes by redirecting queries to the hot node in the database system.

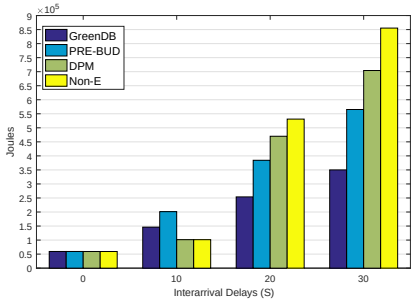
The aforementioned experimental results suggest that heavily loaded conditions (i.e., low inter-arrival delays) prevent GreenDB and PRE-BUD from conserving energy. In contrast, GreenDB and PRE-BUD exhibit good energy efficiency under light and medium workload. In the worst case where the load becomes extremely high, GreenDB will have to increase the number of hot node to suppress high query response time.

4.3.7 Number of Disks per Cold Node

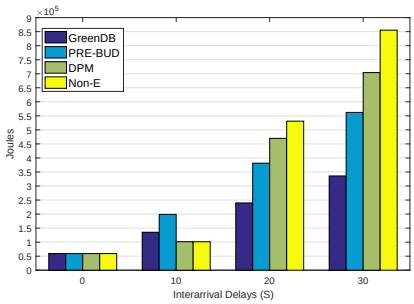
Now we evaluate the capacity impact of cold nodes. In particular, we investigate whether or not increasing the number of disks in each cold node can optimize the energy efficiency of database systems running on clusters. Fig. 4.8 shows the energy consumption of the database system, where the number of disks in each node is set to one, two, and four. In this group of



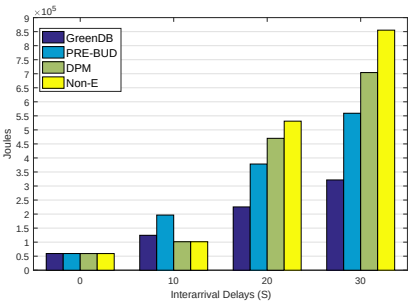
(a) Total energy consumption when the hit rate is : 55%.



(b) Total energy consumption when the hit rate is : 65%.



(c) Total energy consumption when the hit rate is : 75%.



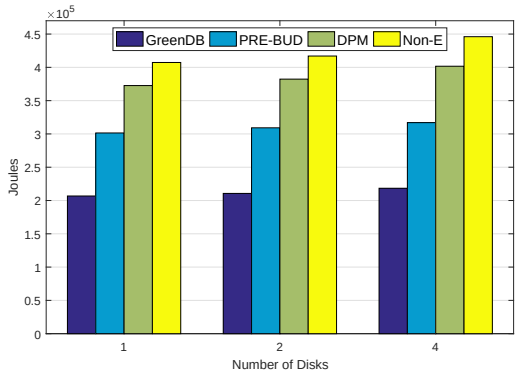
(d) Total energy consumption when the hit rate is : 85%.

Figure 4.8: Total energy consumption for different delay values where the hit rate is : (a) 55%, (b) 65%, (c) 75%, and (d) 85%.

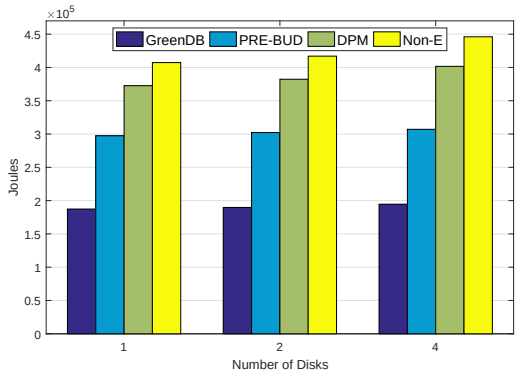
experiments, the data size is fixed at $200K$ records; the hit rate is varied at 60%, 75%, and 90%, respectively.

Not surprisingly, we discover from Fig. 4.9 that increasing the number of disks in each node marginally improves the energy efficiency of GreenDB. For example, let us consider a case where hit rate is 75% (Fig. 4.9(b)). GreenDB reduces energy consumption of Non-E by 54.0% and 56.4% when the number of disks per node is one and four, respectively. If hit rate is 90%, then GreenDB's energy saving is 58.8% and 61.7% for 1-disk and 4-disk cases. The results indicate that adding two extra disks merely increases the energy savings by approximately 2.4% and 2.9% when hit rate is set to 75% and 90%, respectively.

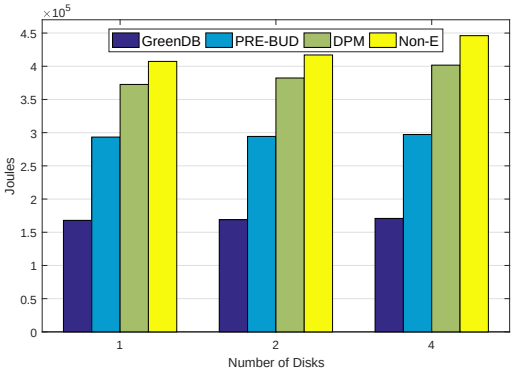
The number of disks per node has little impact on GreenDB's energy efficiency, because each disk's power is fairly small compared with that of the entire node. This experiment suggests that with respect to energy saving, a node-level power manager (e.g., GreenDB) is far more efficient than a disk-level one (e.g., PRE-BUD). The results show the evidence that GreenDB is superior to PRE-BUD in terms of energy efficiency.



(a) Total energy consumption of database clusters when hit rate is fixed at: 60%



(b) Total energy consumption of database clusters when hit rate is fixed at: 75%



(c) Total energy consumption of database clusters when hit rate is fixed at: 90%

Figure 4.9: Total energy consumption of database clusters while the number of data disks of cold node is varied. Hit rate is fixed at: (a) 60%, (b) 75%, (c) 90%

From the perspective of performance, increasing number of of disks in cold node tends to introduce performance bottleneck in hot nodes. One way of solving this problem is to add disks into hot nodes when we expand cold nodes.

4.4 Summary

In this chapter, we proposed *GreenDB* - a simple yet efficient database cluster system. GreenDB is conducive to fetching the most frequently used data tables into hot nodes. We built a prefetching model in conjunction with an energy saving model, which offers guidelines to GreenDB to conserve energy. GreenDB exhibits two salients features. First, GreenDB aggressively keeps cold nodes in the low-power mode in long time periods. Second, GreenDB lowers energy-saving overhead by reducing the number of power-state transitions.

We conducted extensive experiments to quantitatively evaluate the energy efficiency and performance of GreenDB in terms of fetching and managing data tables in database clusters. We compared GreenDB's prefetching strategy with an existing one implemented in Postgresql. Experimental results show that GreenDB conserves the energy consumption of the existing scheme by up to 98.4%. The findings suggest that one may optimize the energy efficiency of GreenDB by configuring table size, hit rates, number of nodes, number of disks, and inter-arrival delays.

Chapter 5

Conclusions and Future Work

In this dissertation, we proposed a simple yet effective toolkit called EDOM facilitating the evaluation and optimization of energy-efficient multicore-based. We investigated the criteria as well as challenges of developing energy efficiency benchmarks for database operations. A benchmarking toolkit was implemented to evaluate energy efficiency of database systems. We evaluated energy-efficiency impacts of multicore processors on database operations. In particular, we applied the benchmarking tool to empirically study energy consumption of cross and outer joins running on multi-core processors. To optimize the number of cores, we developed a multicore manager called *EDOM* to make a good tradeoff between energy efficiency and performance in database systems. The key component of *EDOM* is a memory usage model estimating memory utilization from queries and database characteristics. An appropriate number of cores is determined using the estimated memory usage to alleviate the memory swapping problem - a main driver behind high energy cost in multicore database systems.

In addition, traditional energy-saving techniques for clusters are inadequate for parallel database systems running on clusters. To address this problem, we proposed GreenDB focusing on reducing energy consumption cost of large-scale database systems, which are comprised of multiple nodes or severs. We showed how to develop a parallel database system called *GreenDB* - an energy efficient system running on clusters. At the heart of *GreenDB* is a prefetching and caching mechanism, which fetches hot data from passive nodes into active nodes. We demonstrated that *GreenDB* offers energy savings by turning a large number of passive nodes into the low-power mode, which keeping a small number of active nodes to serve queries.

The chapter is organized as follows: Section 5.1 highlights the main contributions of the dissertation. In section 5.2, we concentrate on some future directions, which are extensions of our past and current research on green computing for high-performance database systems.

5.1 Main Contributions

Energy efficiency techniques is a large research area including energy efficiency profiling, hardware resources management, data placement, workload skewness strategy, and data prefetching strategies. Although tons of research studies have been completed to improve the performance of parallel systems, little attention has been paid to energy-efficient big-data systems. Energy cost turns into a huge fraction of a big data center's maintenance cost. The energy conserving techniques involves multiple factors such as task-scheduling, hardware tuning, system architecture design. To improve the energy efficiency of the big data platforms by systematically considering these factors, my dissertation research investigates techniques to conserve energy consumption while maintaining high performance in database systems.

5.1.1 GreenDB - A New Energy-efficient Database Cluster

GreenDB applies a workload skewness strategy by managing hot nodes coupled with a set of cold nodes in a database cluster. A database cluster governed by GreenDB is divided into a group of cold nodes and a group of hot nodes. The overarching goal is to keep the cold nodes in the low power state (e.g. shutting down) in increased time periods by prefetching popular data tables to hot nodes, while making the hot nodes respond queries accessing popular data tables. There are three motivations behind partitioning nodes into the hot and cold groups. First, our preliminary findings suggest that improving the utilization of an active database node leads to high energy efficiency. Second, placing an entire node into the low-power state is more energy efficient than turning off disks in the node. Third, reducing the number of power-state transitions can lower the power management overhead. My experimental results indicate that GreenDB significantly conserves the energy consumption than the existing solution. The

findings show that the energy efficiency of GreenDB can be optimized by tuning system parameters, including table size, hit rates, number of nodes, number of disks, and inter-arrival delays.

5.1.2 Leveraging Energy Savings Model to Guide Prefetching

To facilitate the replacement strategy in GreenDB, we delve into an energy-cost model projecting energy savings offered by keeping candidate data tables into an array of hot nodes while shutting down under-utilized cold nodes. The energy-cost model provides a guideline for GreenDB to (1) evict unpopular data tables from the hot nodes and (2) fetch popular tables from the cold nodes to improve energy efficiency. The energy-cost model integrates profiling results and the prefetching algorithm to offer us an interface to estimate energy savings provided by data tables in hot nodes. Profiling results take into account of three factors energy consumed on fetching data from cold nodes, energy spent by hot nodes to store prefetched data tables, and energy overhead caused by power-state transitions. Our energy cost model leverages the profiling data of energy consumption to estimate energy savings of offered by prefetched table.

5.1.3 An Asynchronized Prefetching Mechanism

The existing prefetching scheme in PostgreSQL relies on the synchronous replication module to maintain duplicated data between cold and hot nodes. Such a synchronous based prefetching strategy (or SyncPrefetch for short) in PostgreSQL is expensive when it comes to maintaining replicated data on hot nodes. Before committing an update transaction, SyncPrefetch must obtain exclusive locks on all the copies. The transaction may have to send lock requests to cold nodes, waiting for the locks to be granted. During such a potentially long time period, SyncPrefetch continues holding all the other locks. If the cold nodes or communication links fail, the transaction cannot commit until all the nodes storing modified data are recovered the failure. Unlike SyncPrefetch, the prefetching scheme in GreenDB takes the full advantage of asynchronous replications to cut energy cost of fetching hot data from cold nodes. Updated data are initially kept in hot nodes to avoid excessive power-state transitions in cold nodes. When new popular data are about to be transferred from the cold nodes to the hot ones, the

update data will be synchronized back to the cold nodes. GreenDB's synchronize replicated data between hot and cold nodes in a lazy manner, thereby improving system energy efficiency.

5.1.4 An Evaluation and Optimization Toolkit

I designed a toolkit called EDOM facilitating the evaluation and optimization of energy-efficient multicore-based database systems. I built a benchmarking toolkit, which is comprised of three parts, namely, a configuration module, a test driver, and a power monitor. The workload generator facilitates the configurations of the PostgreSQL database system. We leverage this generator to set up tables and populate data records into the database. The test driver automatically issues operations to the database system in accordance to access patterns created by the workload generator. The power monitor keeps track of energy efficiency and performance of the multicore database system processing the operations driven by the test driver. I evaluated energy-efficiency impacts of multicore processors on database operations. In particular, I applied the benchmarking tool to empirically study energy consumption of cross and outer joins running on multi-core processors. My benchmarking experiments show that the multicore and CPU utilizations have significant impacts on energy efficiency.

5.1.5 A Memory Usage Model for Queries and Database Characteristics

In addition to my research of EDOM, I developed a multicore manager to optimize the number of cores, thereby making the best tradeoff between performance and energy efficiency in multicore database servers. The multicore manager aims to decide an optimal number of cores that meets the resource needs of heavy workload, where main memory becomes scarce resources. An appropriate number of cores is determined by using the estimated memory usage to avert unnecessary memory swapping, which induces high energy consumption overhead. This goal is achieved by alleviating the memory swapping problem through the decisions on the most appropriate number of cores.

5.1.6 A Multicore Manager Enhancing Energy Efficiency

At the heart of the multicore manager is a memory usage model that estimates memory utilization from queries and database characteristics (e.g., query types, data size, and processing time). My empirical study reveals that the memory usage imposes a significant impact on the energy efficiency of database systems, heavily utilized main memory adversely slows downs the performance of multiple cores; as a result, an increasing number of queries cannot be processed in a timely manner, thereby pushing the power consumption at an unacceptably high level. To address the problem of heavy workload coupled with scarce memory resources, the memory usage estimator estimates the memory usage according to the four operation factors, namely, database query type, the number of tables, the number of records and the record size. Given such information the memory usage estimator applies a mathematical model to project memory usage, which is used in the multicore calculator to determine the number of the mulit-cores.

5.2 Future Work

As future research directions, I plan to extend this dissertation research by focusing on statistical prediction strategies (see Section 5.2.1), performance tuning in database systems (see Section 5.2.2), data placement (see Section 5.2.3), NoSQL database systems (see Section 5.2.4), energy-efficient big-data processing systems (see Section 5.2.5), in-memory data mining (see Section 5.2.6), and high-performance computing (see Section 5.2.7).

5.2.1 Statistical Prediction Strategies for Energy-Efficient Database Systems

Since multiple factors (i.e., CPU, memory, data size, and operation type) will influence the energy efficiency of database systems, I plan to investigate a novel and sophisticated strategy that can take use the of statistical or machine learning algorithms to provide accurate hardware management to improve the energy efficiency of database systems.

5.2.2 Dynamic Tuning of Database Systems

In the GreenDB project, I adopted hot nodes to cache the most popular data tables so that we can switch cold nodes to the low-power state. Although it improves the energy efficiency of the database clusters, its requires the prior access patterns and log traces to adjust the database clusters. I intend to develop a new real-time analysis mechanism to tune the database clusters. One way is to leverage big-data techniques and machine learning algorithms to analyze and predict resource requirements to meet the needs of queries or application-level request.

5.2.3 Optimizing Data Placement in Database Clusters

Big-data applications tend to have a huge amount of transferred data. Since the transferred data may be moved from node to node, data movement has a significant impact on the overall energy-efficiency whole system due to its long period active status without any chance to switch it into low power state. To optimize the data placement which will effectively reduce the unnecessary data movement, I will propose a predictive model to store the massive amount of data in the most appropriate nodes and move data without compromising the performance of applications running on local nodes in data clusters. The new model largely depends on data analysis techniques, data distribution, the amount of data, data access patterns, and network traffic.

5.2.4 Improving Energy Efficiency of NoSQL Database Systems

With the development of the Internet and cloud computing, there is a pressing need to store and process big data on clusters. This issue is addressed by NoSQL databases, which increasingly become popular. Traditional SQL databases have a performance bottleneck due to following four overhead components, namely, logging, locking, latching, and buffer management. Although NoSQL systems are likely to be deliver good data read and write performance, a majority of NoSQL systems are equipped with hard drives, where a buffer pool is maintained. I plan to improve the energy efficiency in NoSQL systems by investigating techniques to reduce

unnecessary I/Os, which in turn will offer an increasing number of chances to switch disks into the low-power state.

5.2.5 Improve the Energy Efficiency within Big-data Platforms

Increasing demands of big data applications have led researchers and practitioners to turn to in-memory computing to speed up processing. For instance, the Apache Spark framework stores intermediate results in memory to deliver good performance on iterative machine learning and interactive data analysis tasks. Recent studies indicate that Spark workloads have a high number of disk accesses per second. The goal of my future investigation is to examine disk accesses patterns of Spark workloads, followed by designing a middle-aware layer optimizing the number of disk accesses. In doing so, my proposed mechanism will cut back energy spending on unnecessary disk accesses.

5.2.6 In-Memory Data Placement and Computing for Data Mining Techniques

As data mining techniques have been widely used and developed in this decade, one main challenge in data mining is large-scale parallel processing of a massive amount data on clusters (e.g., Hadoop clusters and Spark clusters). Data volumes in and data warehouses are dramatically increasing; unfortunately, computing platforms can't keep the pace with the increased data size. Most of existing data mining algorithms are computational intensive, requiring data to be resident in main memory to speedup process. My goal in this research direction is to investigate data placement strategies to prefetching data from hard drives of a database cluster to facilitate novel in-memory database computing, which is expected to improve the overall performance of data mining algorithms by optimizing I/O subsystems. My in-memory data placement techniques intend to reduce data movement while offering quick data accesses.

5.2.7 High-Performance Data Accesses in Big Data Platforms

I will propose high-performance data accesses in the realm of "NoSQL" databases running on big-data computing platforms. I was motivated to address this issue, because the data access

time directly impacts the number performance of answering queries. Therefore, I intend to deploy new techniques to facilitate fast data accesses, which will improve the query performance in big data platforms.

References

[1] Wd5000aaks specification. http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701277.pdf.

[2] Accenture and WSP. Cloud computing and sustainability: The environmental benefits of moving to the cloud. Technical report, Accenture, 2010.

[3] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. *SIGOPS Oper. Syst. Rev.*, 45(3):53–57, Jan. 2012.

[4] M. Al Assaf, X. Jiang, M. Abid, and X. Qin. Eco-storage: A hybrid storage system with energy-efficient informed prefetching. *Journal of Signal Processing Systems*, 72(3):165–180, 2013.

[5] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012. ¡ce:title¿Special Section: Energy efficiency in large-scale distributed systems¡/ce:title¿.

[6] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[7] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on*

Cluster, Cloud and Grid Computing, CCGRID '10, pages 826–831, Washington, DC, USA, 2010. IEEE Computer Society.

[8] Y. Chehadeh, A. R. Hurson, and L. L. Miller. Energy-efficient indexing on a broadcast channel in a mobile database access system. In *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*, pages 368–374. IEEE, 2000.

[9] F. Chen, J.-G. Schneider, Y. Yang, J. Grundy, and Q. He. An energy consumption model and analysis tool for cloud computing environments. In *Proceedings of the First International Workshop on Green and Sustainable Software*, pages 45–50. IEEE Press, 2012.

[10] D. Chiu, C. Stewart, and B. McManus. Electric grid balancing through lowcost workload migration. *SIGMETRICS Perform. Eval. Rev.*, 40(3):48–52, Jan. 2012.

[11] G. Clayton, K. Pike, R. Nash, D. Hutton, and C. Rogers. Improving the efficiency of testing database functionality through statistical involvement. *Trials*, 16(Suppl 2):P30, 2015.

[12] A. T. Clements, M. F. Kaashoek, N. Zeldovich, R. T. Morris, and E. Kohler. The scalable commutativity rule: Designing scalable software for multicore processors. *ACM Transactions on Computer Systems (TOCS)*, 32(4):10, 2015.

[13] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing '02, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[14] G. Cook. How clean is your cloud? Technical report, Greenpeace International, April 2012.

[15] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Googles globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[16] W. Deng, F. Liu, H. Jin, C. Wu, and X. Liu. Multigreen: Cost-minimizing multi-source datacenter power supply with online control. In *Proceedings of the Fourth International Conference on Future Energy Systems*, e-Energy '13, pages 149–160, New York, NY, USA, 2013. ACM.

[17] T. Endo and G. Jin. Software technologies coping with memory hierarchy of gpgpu clusters for stencil computations. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, pages 132–139. IEEE, 2014.

[18] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. Sap hana database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51, 2012.

[19] G. Graefe. New algorithms for join and grouping operations. *Computer Science-Research and Development*, 27(1):3–27, 2012.

[20] G. Graefe, H. Volos, H. Kimura, H. Kuno, J. Tucek, M. Lillibridge, and A. Veitch. In-memory performance for big data. *Proceedings of the VLDB Endowment*, 8(1):37–48, 2014.

[21] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, Dec. 2008.

[22] S. K. S. Gupta, A. Banerjee, Z. Abbasi, G. Varsamopoulos, M. Jonas, J. Ferguson, R. R. Gilbert, and T. Mukherjee. Gdcsim: A simulator for green data center design and analysis. *ACM Trans. Model. Comput. Simul.*, 24(1):3:1–3:27, Jan. 2014.

[23] A. Hammadi and L. Mhamdi. Review: A survey on architectures and energy efficiency in data center networks. *Comput. Commun.*, 40:1–21, Mar. 2014.

[24] S. Harizopoulos, M. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. *arXiv preprint arXiv:0909.1784*, 2009.

[25] J. Huang, F. Zhang, X. Qin, and C. Xie. Exploiting redundancies and deferred writes to conserve energy in erasure-coded storage clusters. *Trans. Storage*, 9(2):4:1–4:29, July 2013.

[26] I. E. Insights. Annual it spending by western european utilities to reach 12.7 billion by 2017, Aug 2013.

[27] P. Jones. Industry census 2012: Emerging data center markets. `https://www.datacenterdynamics.com/blogs/industry-census-2012-emerging-data-center-markets`, October 2012.

[28] J. Kim, M. Ruggiero, and D. Atienza. Free cooling-aware dynamic power management for green datacenters. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 140–146. IEEE, 2012.

[29] D. Kliazovich, P. Bouvry, and S. U. Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[30] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B.-g. Chun, L. Huang, P. Maniatis, M. Naik, and Y. Paek. Mantis: Efficient predictions of execution time, energy usage, memory usage and network usage on smart mobile devices. *Mobile Computing, IEEE Transactions on*, 14(10):2059–2072, 2015.

[31] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis. Towards energy-efficient database cluster design. *Proceedings of the VLDB Endowment*, 5(11):1684–1695, 2012.

[32] Y. Lee and A. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[33] Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *J. Supercomput.*, 60(2):268–280, May 2012.

[34] A. A. Lima, C. Furtado, P. Valduriez, and M. Mattoso. Parallel olap query processing in database clusters with data replication. *Distributed and Parallel Databases*, 25(1-2):97–123, 2009.

[35] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, 21(5):1378–1391, 2013.

[36] H. Ltaief, P. Luszczek, and J. Dongarra. Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency. *Computer Science-Research and Development*, 27(4):277–287, 2012.

[37] A. Manzanares, X. Qin, X. Ruan, and S. Yin. Pre-bud: Prefetching for energy-efficient parallel i/o systems with buffer disks. *ACM Transactions on Storage (TOS)*, 7(1):3, 2011.

[38] R. Miller. Facebook's $1 billion data center network. `http://www.datacenterknowledge.com/archives/2012/02/02/facebooks-1-billion-data-center-network/`, February 2012.

[39] M. P. Mills. The cloud begins with coal: Big data, big networks, big infrastructure, and big power. `http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf?c761ac&c761ac`, August 2013.

[40] A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das. Coordinated power management of voltage islands in cmps. *SIGMETRICS Perform. Eval. Rev.*, 38(1):359–360, June 2010.

[41] J. A. Mullins, A. T. Cooney, B. T. Butler, G. V. Hallissey, D. A. Barrett, J. Carmody, P. O'keeffe, P. J. Healy, L. O'mahony, P. Sheehan, et al. Data center energy manager for monitoring power usage in a data storage environment having a power monitor and a monitor module for correlating associative information associated with power consumption, June 17 2014. US Patent 8,756,441.

[42] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys (CSUR)*, 46(4):47, 2014.

[43] C. Patel and P. Ranganathan. Enterprise power and cooling. *ASPLOS tutorial*, 2006.

[44] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *Distributed Computing*, pages 315–329. Springer, 2000.

[45] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, pages 68–78, New York, NY, USA, 2004. ACM.

[46] M. Poess and R. O. Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. *Proceedings of the VLDB Endowment*, 1(2):1229–1240, 2008.

[47] R. Ramakrishnan. Database management systems . pdf. 2000.

[48] S. Ren and Y. He. Coca: Online distributed resource management for cost minimization and carbon neutrality in data centers. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 39:1–39:12, New York, NY, USA, 2013. ACM.

[49] W. Rodiger, T. Muhlbauer, P. Unterbrunner, A. Reiser, A. Kemper, and T. Neumann. Locality-sensitive operators for parallel main-memory database clusters. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 592–603. IEEE, 2014.

[50] H. Saxena and K. Salem. Edgex: Edge replication for web applications. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 1041–1044. IEEE, 2015.

[51] D. Schall and T. Härder. Towards an energy-proportional storage system using a cluster of wimpy nodes. In *BTW*, pages 311–325. Citeseer, 2013.

[52] D. Schall and T. Härder. Approximating an energy-proportional dbms by a dynamic cluster of nodes. In *Database Systems for Advanced Applications*, pages 297–311. Springer, 2014.

[53] M. Sharifi, H. Salimi, and M. Najafzadeh. Power-efficient distributed scheduling of virtual machines using workload-aware consolidation techniques. *J. Supercomput.*, 61(1):46–66, July 2012.

[54] J. Shuja, K. Bilal, S. A. Madani, M. Othman, R. Ranjan, P. Balaji, and S. U. Khan. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Systems Journal*, 10(2):507–519, 2016.

[55] P. Thibodeau. Data centers use 2% of u.s. energy, below forecast, August 2011.

[56] W. Tian, Q. Xiong, and J. Cao. An online parallel scheduling method with application to energy-efficiency in cloud computing. *J. Supercomput.*, 66(3):1773–1790, Dec. 2013.

[57] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 231–242. ACM, 2010.

[58] F. Urbano and H. Dettki. Storing tracking data in an advanced database platform (postgresql). In *Spatial Database for GPS Wildlife Tracking Data*, pages 9–24. Springer, 2014.

[59] S. D. Viglas. A comparative study of implementation techniques for query processing in multicore systems. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):3–15, 2014.

[60] J. Whitney and J. Kennedy. Is cloud computing always greener? Technical report, Natural Resources Defense Council, October 2012.

[61] M. G. Xavier, I. C. De Oliveira, R. D. Dos Passos, and C. A. De Rose. Towards better manageability of database clusters on cloud computing platforms. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 366–367. ACM, 2014.

[62] Z. Xu, Y.-C. Tu, and X. Wang. Pet: reducing database energy cost via query optimization. *Proceedings of the VLDB Endowment*, 5(12):1954–1957, 2012.

[63] M. Zapater, J. L. Ayala, and J. M. Moya. Leveraging heterogeneity for energy minimization in data centers. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 752–757, Washington, DC, USA, 2012. IEEE Computer Society.

[64] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, R. Y. Wang, et al. Modeling hard-disk power consumption. In *FAST*, volume 3, pages 217–230, 2003.

[65] W. Zheng, S. Tu, E. Kohler, and B. Liskov. Fast databases with fast durability and recovery through multicore parallelism. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 465–477, 2014.

[66] Y. Zhou, S. Taneja, X. Qin, W.-S. Ku, and J. Zhang. Edom: Improving energy efficiency of database operations on multicore servers. *Future Generation Computer Systems*, 2017.

[67] Z. Zong, A. Manzanares, X. Ruan, and X. Qin. Ead and pebd: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. *Computers, IEEE Transactions on*, 60(3):360–374, 2011.

1