Future Generation Computer Systems 105 (2020) 1002–1015

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

EDOM: Improving energy efficiency of database operations on multicore servers

Yi Zhou [a], Shubbhi Taneja [a,*], Xiao Qin [a], Wei-Shinn Ku [a], Jifu Zhang [b]

[a] Department of Computer Science and Software Engineering, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347, United States

[b] School of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan 030024, China

article info

Article history:
Received 16 May 2016
Received in revised form
3 October 2016
Accepted 24 February 2017
Available online 6 March 2017

Keywords:
Energy efficiency
Database operations
Multicore processors
Benchmarks
Data centers
Database applications

abstract

In this paper, we propose a toolkit called *EDOM* facilitating the evaluation and optimization of energy-efficient multicore-based database systems. The 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 describe the criteria and challenges of building an energy efficiency benchmark for databases on multicore servers. We build a benchmarking toolkit, which is comprised a configuration module, a test driver, and a power monitor. We develop a multicore manager to optimize the number of cores, thereby making good 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. An appropriate number of cores is determined using the estimated memory usage to avert unnecessary memory swapping. 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 utilizations have significant impacts on energy efficiency. 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.

1. Introduction

Because multicore processors have been widely utilized by various database applications (see, for example, [1]), evaluating energy efficiency of database applications running on multicore systems becomes indispensable and strategic components of building green data centers.

High energy efficiency is of importance for reducing operating cost of data centers, where database applications are running on multicore servers [2]. 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.

The following four motivations make energy-efficiency benchmarking tools and multicore managers for database operations desirable and achievable.

(1) Rising energy costs in large data centers are driving an agenda for energy-efficient database systems.
(2) The lack of study on the energy efficiency of database operations (e.g., cross and outer joins) running on multicore servers.
(3) The pressing need of benchmarking tools for energy-efficient database systems.
(4) The growing importance of improving energy efficiency of database systems through multicore management.

Motivation 1. Energy cost is one of the significant components of operational costs in data center environments [3]. Evidence shows that a data center containing 1000 racks consumes 10 MW

* Corresponding author.
E-mail addresses: yzz0074@auburn.edu (Y. Zhou), shubbhi@auburn.edu (S. Taneja), xqin@auburn.edu (X. Qin), weishinn@auburn.edu (W.-S. Ku), zjf@tyust.edu.cn (J. Zhang).

http://dx.doi.org/10.1016/j.future.2017.02.043
0167-739X/

total power per year [4]. 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 [5]. Unfortunately, little attention has been paid to energy efficiency improvement of database applications, the popularity of which is rapidly growing. The lack of energy-efficient database systems motivates us to investigate energy efficiency of modern database systems.

Motivation 2. Computer architecture enters a new era of multi-core structures, which become a standard computing platform for a wide range of application domains including database systems [6]. 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 [7], 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.

Motivation 3. 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, JouleSort is used to evaluate the energy efficiency of clusters [8]. 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.

Motivation 4. 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 [9]. 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.

Contributions. We make the following six contributions in this study.

(1) The criteria and challenges of developing energy efficiency benchmarks for database operations.
(2) A toolkit called *EDOM* facilitating the evaluation and optimization of energy-efficient multicore-based database systems.
(3) A benchmarking tool in *EDOM* to evaluate energy-efficient database systems.
(4) An analysis of energy-efficiency impacts of multicore processors on database operations.
(5) An energy-aware multicore manager—a core component in *EDOM*.
(6) The evaluation of EDOM in terms of energy consumption and performance.

We investigate the criteria as well as challenges of developing energy efficiency benchmarks for database operations. Such criteria and challenges provide a general guideline for the design of our energy-efficiency benchmarking tool.

We develop a benchmarking toolkit to evaluate energy efficiency of database systems. The toolkit is comprised of three modules, including a workload generator, 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 evaluate energy-efficiency impacts of multicore processors on database operations. In particular, we apply the benchmarking tool to empirically study energy consumption of cross and outer joins running on multi-core processors. Our benchmarking experiments show that the multicore and CPU utilizations have significant impacts on energy efficiency. For example, we show that multicore processors are more energy efficient than the non-multicore counterparts; the cross and outer join operations have remarkable difference in energy consumption; and the indexing technique improves energy efficiency of the database system.

To optimize the number of cores, we develop a multicore manager – *EDOM* – that makes 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, which is a main driver for high energy cost in multicore database systems.

We make use of the benchmarking toolkit to evaluate EDOM in terms of energy consumption and performance. Our extensive experiments show that EDOM provides a simple yet powerful solution for improving energy efficiency of database applications running on multicore servers.

Paper organization. The rest of the paper is organized as follows. Section 2 summarizes the related work. In Section 3, 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 4. In Section 5, the multicore manager is proposed and the algorithm of the multicore manager is stated. Section 6 provides an investigation in operations regarding energy and performance efficiency, and an in depth analysis is presented after that. Finally, Section 7 shows conclusions including the contributions of this research along with our future research directions.

2. Related work

Energy-efficient clusters are becoming increasingly popular in large-scale data centers [10] 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. For example, Zong et al. proposed two energy-efficient duplication-based scheduling algorithms called EAD and PEBD to achieve the goal of optimizing both performance and energy efficiency in clusters [5]. Rather than investigating energy-efficient clusters, in this study we focus on energy-efficient database management systems.

There is a demanding need to develop energy-efficient database management systems (DBMS). 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 [11]. Lang et al. analyzed a number of important parameters related to the design of energy-efficient DBMS [7]. A framework built by Lang et al. optimizes queries by considering both performance and energy consumption as optimization criteria [7].

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 [11]. Schall and Härder developed a distributed DBMS – WattDB – to make dynamical

configurations to satisfy performance demands while conserving energy consumption on clusters [12]. Xu et al. implemented PET—an energy-aware query optimization framework that is an extension of the PostgreSQL kernel [13]. 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, energy consumption models have been developed for computing systems [14]. 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 [15]. For instance, by focusing on the energy cost estimation of query plans, Xu and Tu build a series of models for energy estimation of individual relational operators based on their resource consumption patterns [16]. Different from the existing energy consumption models, our models built in EDOM pay attention to modeling energy consumption of DBMS operations.

3. 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., [7]) summarized in Section 2 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.

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 [17]. 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 [18]. 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).

Energy 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.

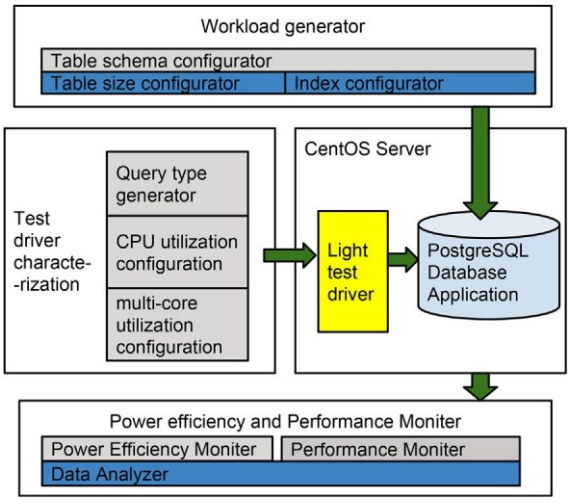


Fig. 1. The framework of the energy efficiency benchmarking toolkit, which consists of a configuration module, a test driver, and a power-performance monitor.

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 [19]; on the other hand, improving system performance is a crucial aspect of metrics to evaluate overall database efficiency [20]. 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 [21]. 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.

Optimization for multicore-based database systems. It is challenging to optimize database systems running on multicore processors [22]. 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 [23], 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 [24]. 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.

4. 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. 1 consists of three components, namely, the configuration module, the test driver, and the power-performance monitor.

Table 1
Testbed configurations.

	OptiPlex 3020 MT/SFF technical specifications
CPU	Intel 4th Core i5-4570 Quad Core@3.20 GHz
Memory	4 GB Non-ECC 1600 MHz DDR3 SDRAM
Hard drives	Seagate KC47-500 GB 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

4.1. 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. 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.

4.2. 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.

4.3. 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 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.

4.4. 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 [25]. ProstgreSQL is capable of processing workloads of small-scale applications as well as large Web-based applications. 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 1 shows the database server specification.

Table 2
Power meter specifications.

	TS-836A power meter specifications
Measurement of consumption	0.00–9999.99 kWh
Voltage display range	0–9999 V
Current range	0.000–15.000 A
Frequency display	0–9999 Hz
Wattage display (Watts)	0–1800 W

4.5. 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 [26]. 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 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.

5. 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 5.1). We show the algorithm of the multicore manager in Section 5.2. The energy efficiency of a database system governed by the multicore manager is evaluated in Section 5.3.

5.1. 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 [27]. 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 [28].

Our empirical study reveals that the memory usage imposes a significant impact on the energy efficiency of database systems. For example, Figs. 2, 5(a), and 6(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. Please refer to Section 6.1.1 for a detailed analysis on the impact of memory utilization on system energy efficiency.

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

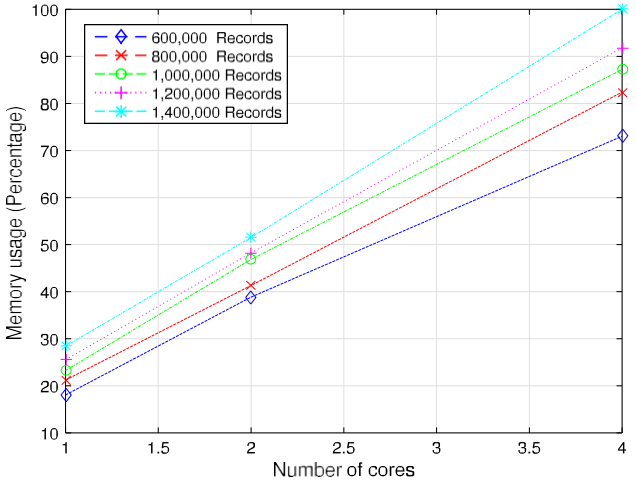


Fig. 2. The impact of the number of cores on memory usage of a multicore-based database system.

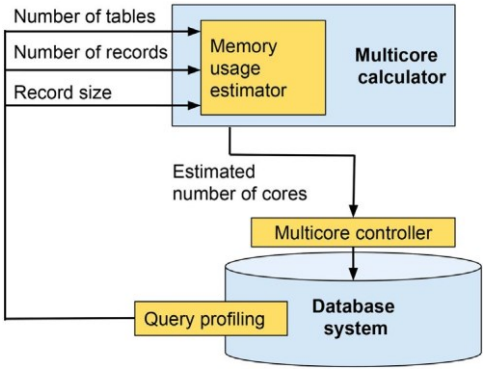


Fig. 3. The framework of the memory usage estimator.

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 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 memory usage estimator applies a mathematical model to project memory usage, which is used in the multicore calculator to determine the number of the multicores. The multicore manager algorithm detailed in Section 5.2 incorporates a multicore calculator (see Fig. 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.

5.2. 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 5.1) 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()*

Input:
  number of tables t
  number of records r
  record size s
Output:
  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()*

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

1: $C_{opt} = \lfloor \frac{C_{min}+C_{max}}{2} \rfloor$
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).

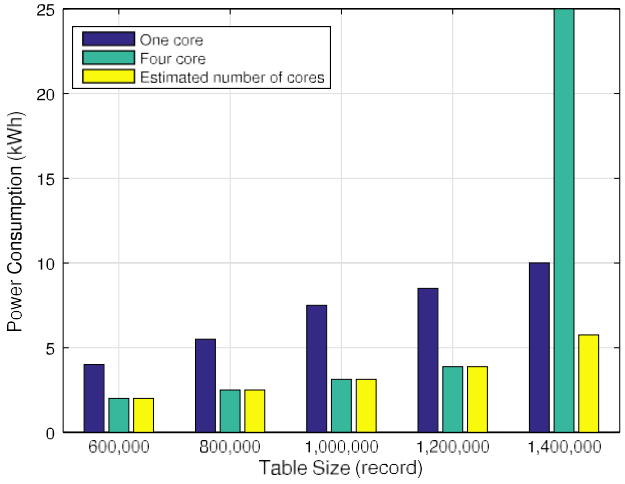


Fig. 4. The validation of energy consumption governed by our multicore manager.

We demonstrate in the next subsection (i.e., Section 5.3) 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.

5.3. Energy efficiency of the multicore manager

We design a group of experiments to study the energy efficiency of the multicore manager. 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. 4 shows the energy consumption of the multicore-based database system governed by our multicore manager; Fig. 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. 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).

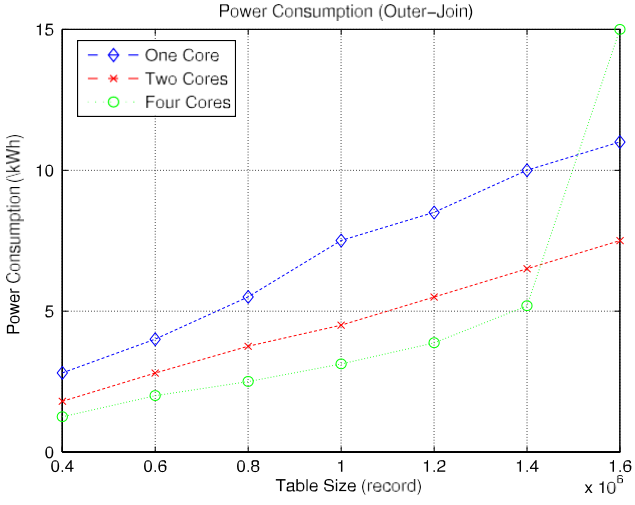
We conclude that the multicore manager in our *EDOM* is conducive to deciding the number of cores needed to optimize the energy efficiency of multicore-based database systems where main memory becomes a scarce resource.

6. Experimental results

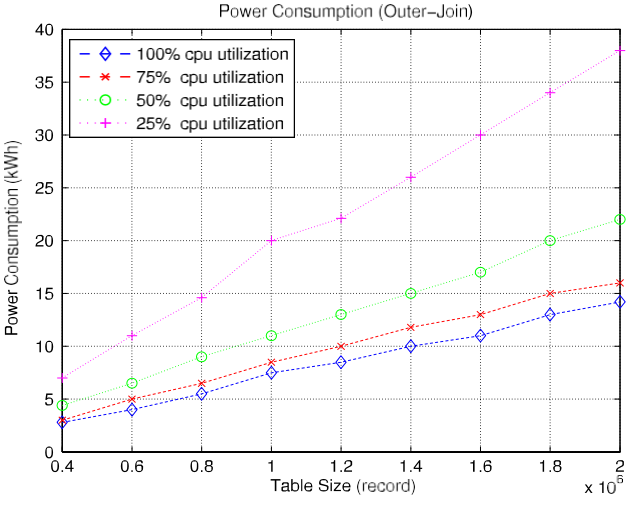
We have conducted extensive experiments to demonstrate the usage and effectiveness of EDOM. We apply the developed EDOM to evaluate the energy efficiency and performance of a multicore-based database server system. In this part of the study, we first investigate the impacts of CPU utilization and multicore 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.

6.1. 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.

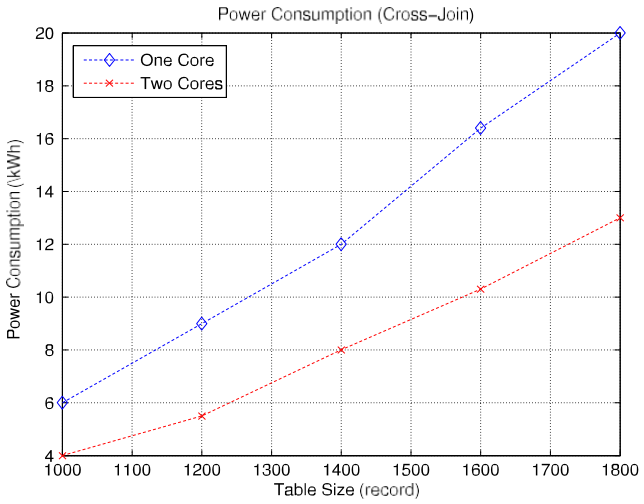


(a) The impact of multicores on power consumption with the Outer-Join operation.

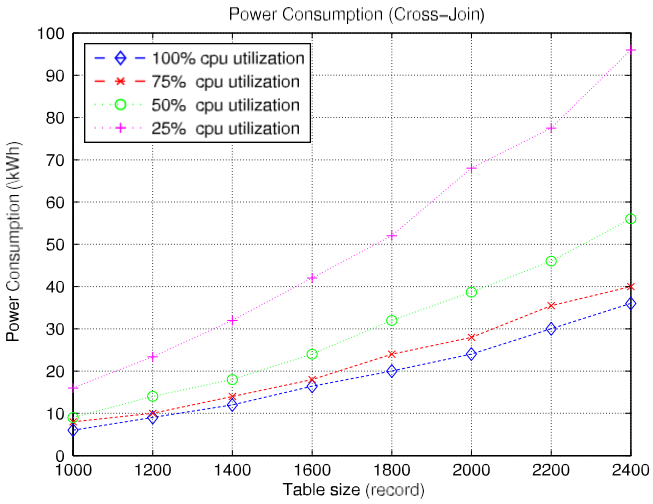


(b) The impact of CPU utilization on power consumption with the Outer-Join operation.

Fig. 5. Power consumption profiling of query: Outer-Join.



(a) The impact of Multicores on power consumption with the Cross-Join operation.



(b) The impact of CPU utilization on power consumption with the Cross-Join operation.

Fig. 6. Power consumption profiling of query: Cross-Join.

6.1.1. 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. 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. 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., 4 GB) 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. 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. 5(a).

6.1.2. 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 6.1.1, in this group of experiments a fixed number of cross-join queries are executed while varying the table size and number of multicores.

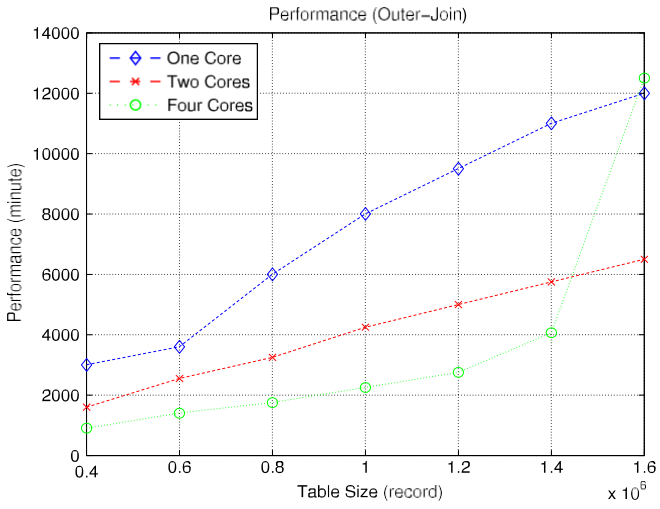
We observe that various types of database operations have different impacts on energy efficiency. Nevertheless, the power consumption trend shown in Fig. 6(a) is similar to that of Fig. 5(a); thus, regardless of the number of cores, energy consumption of the database system goes up when the table size 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. 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. 5(a).

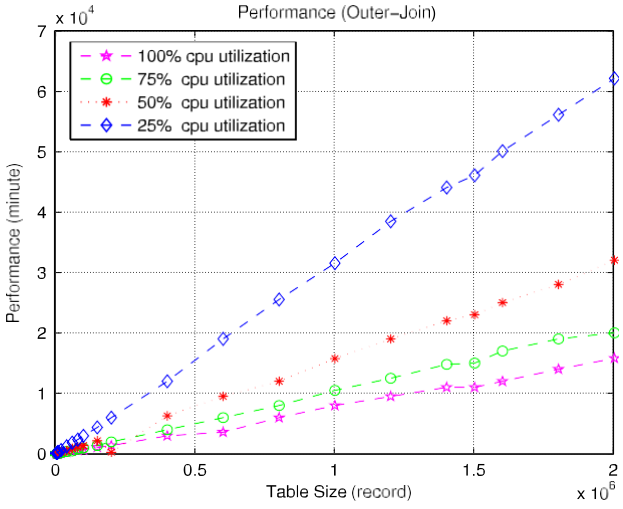
The energy consumption of the three-core and four-core cases are not plotted in Fig. 6(a), because the system's main memory is so heavily utilized that multiple cores are unable to process any query. For example, the three-core system exhibits excessive long response times even when the table size is as small as 1000. We conclude that the memory resource becomes a performance bottleneck of the multicore-based database system. Moreover, compared with the outer-join operations, the energy behavior of cross-join queries are more sensitive to main memory capacity of the system.

Fig. 6(b) shares similar energy consumption patterns as those of Fig. 5(b) (Section 6.1.1). Thus, the system's energy efficiency can be significantly improved by pushing CPU utilization up to 100%.

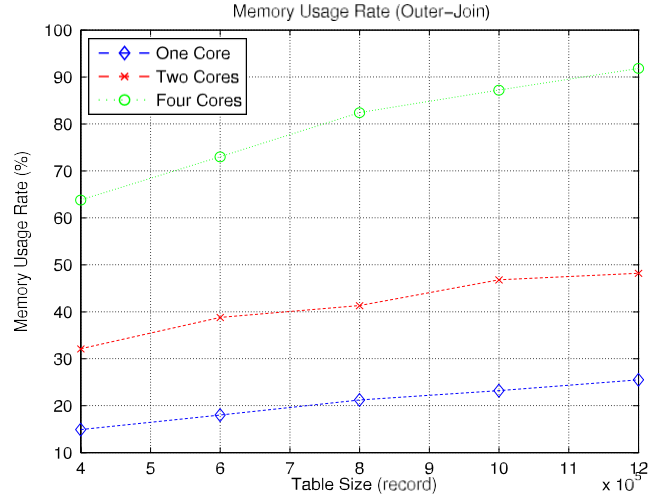
Fig. 6(b) confirms a trend illustrated in Fig. 6(a) that increasing table size leads to a high energy-consumption level. Compared



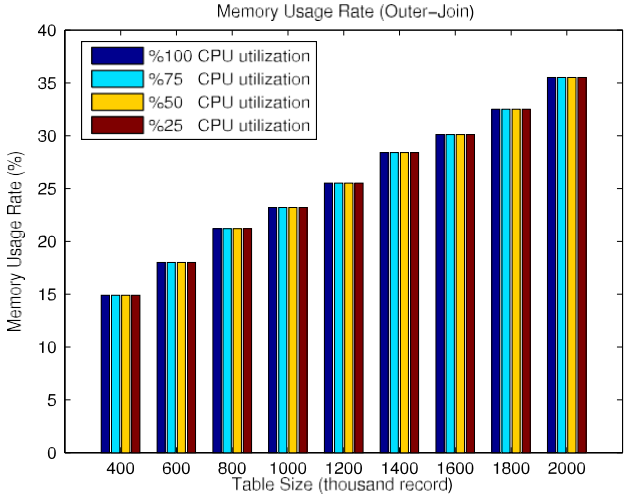
(a) The impact of multicores on time consumption with the Outer-Join operation.



(b) The impact of CPU utilization on time consumption with the Outer-Join operation.



(c) The impact of multicores on memory usage with the Outer-Join operation.



(d) The impact of CPU utilization on memory usage with the Outer-Join operation.

Fig. 7. Performance profiling of Outer-Join under different CPU utilization and multicores.

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.

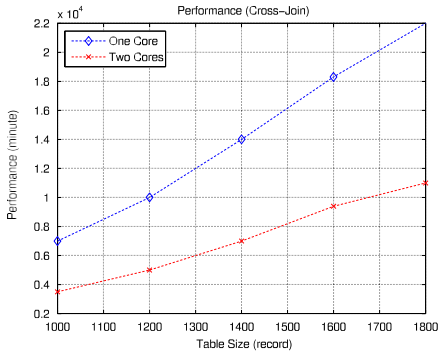
An insightful conclusion drawn from this group of experiments is that we can improve the energy efficiency of database systems by making a full usage of multicore processors in servers.

6.1.3. *Performance of the outer-join operation*

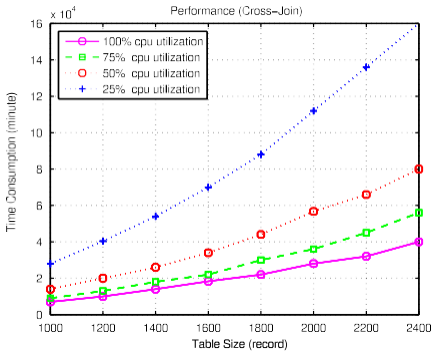
Now we study performance of the outer-join operations. Fig. 7(a) shows the performance as a function of the number of table size in the one-core, two-core, and four-core cases. The results reveal that the outer-join operations exhibit better performance in the four-core configuration than in the other two cases. The four cores reduce the execution time of performing the outer-join queries, thereby improving system energy efficiency (see also Fig. 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. 7(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. 7(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.

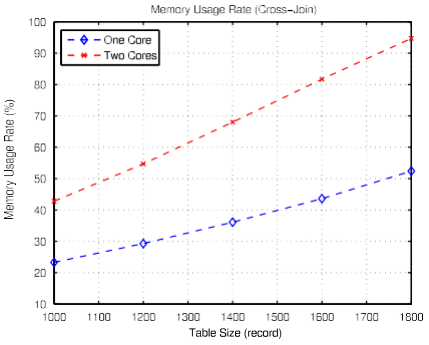
We investigate the memory usage under various table sizes and number of cores. Fig. 7(c) indicates that regardless of the number of cores, increasing the table size slightly drives the memory usage up. Compared with the memory usage in the one-core and two-core systems, memory usage of the four-core system is more sensitive to the page size. For example, when we increase the



(a) The impact of multicores on time consumption with the Cross-Join operation.

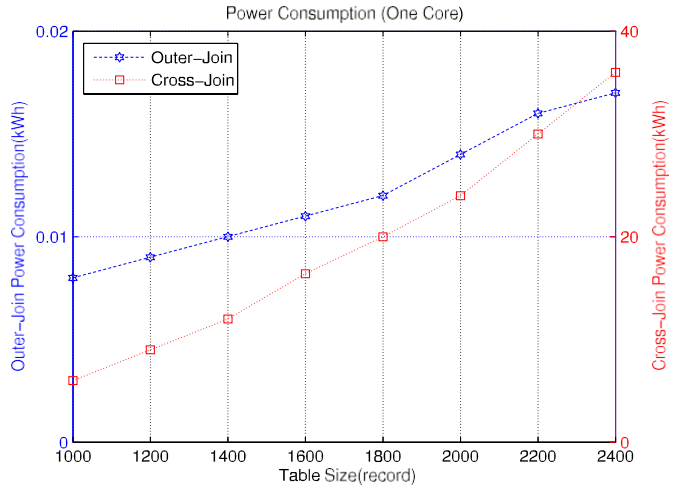


(b) The impact of CPU utilization on time consumption with the Cross-Join operation.

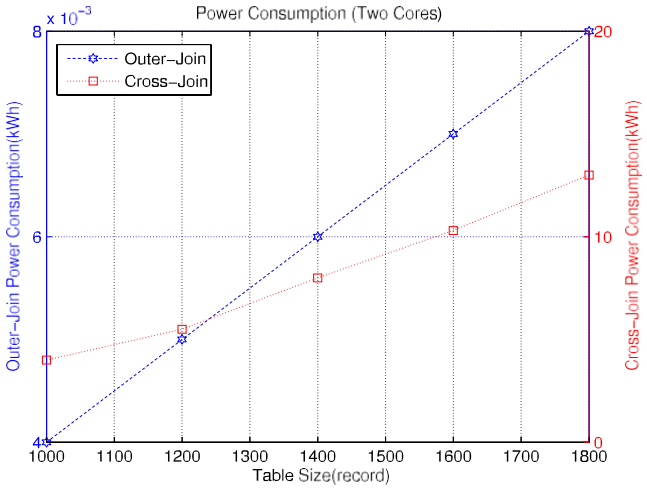


(c) The impact of multicores on memory usage with the Cross-Join operation.

Fig. 8. Performance profiling of Cross-Join under different CPU utilization and multicores.



(a) Power consumption comparison between Outer-Join and Cross-Join in the single-core case.



(b) Power consumption comparison between Outer-Join and Cross-Join in the two-core case.

Fig. 9. Power consumption comparison between outer-join and cross-join in multicore systems.

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. 7(d) 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.

6.1.4. 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. 8(a) shows a similar performance trend as that plotted in Fig. 7(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. 8(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. 7(b).

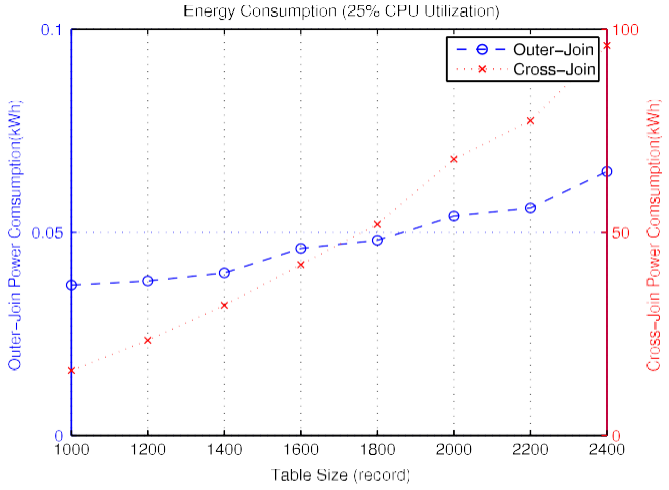
Fig. 8(c) reveals the memory usage of cross-join operations under various table size and the number of cores. Fig. 8(c) shows that the memory usage increases almost linearly with the increasing table size. We observe that compared with the memory usage of the outer-join operations, the memory usage of cross-join operations is more sensitive to the table size (see also Fig. 7(c) and Fig. 8(c)). More detailed comparison between outer-join and cross-join operations can be found in Section 6.2.

6.2. Outer-join vs. cross-join operations

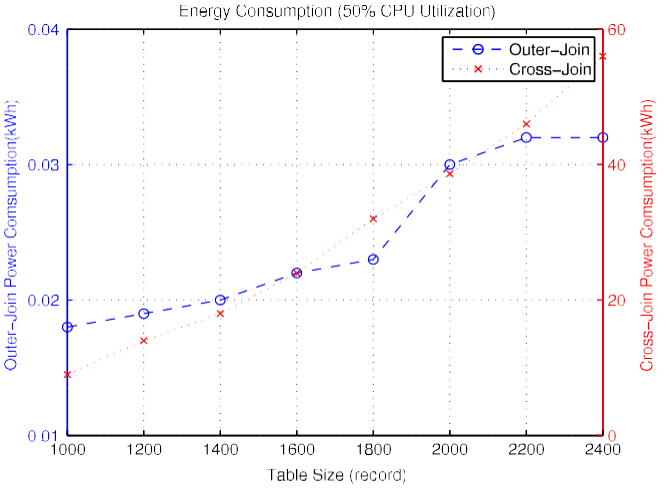
Now we compare the energy behaviors between outer-join and cross-join operations in multicore database systems.

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

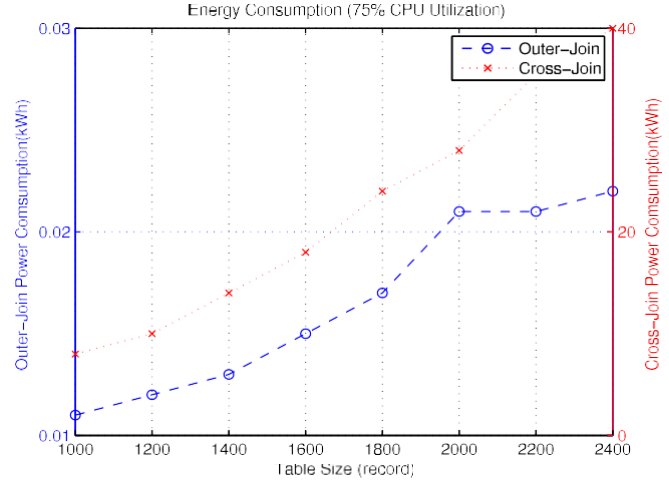
We pay particular attention to the impact of multicores on the outer-join and cross-join queries under the changing table size. The results plotted in Fig. 9(a) and (b) indicate that the query types have significant impacts on energy efficiency. For example, the energy consumption of outer-join is only 0.047% and 0.078% of that of cross-join when table size is set to 2400 and 1800, respectively (see Fig. 9(a) and (b)). Thus, the energy consumption of cross-join queries is 1000–10 000 times higher than that of outer-join queries. This energy consumption trend is attributed to two



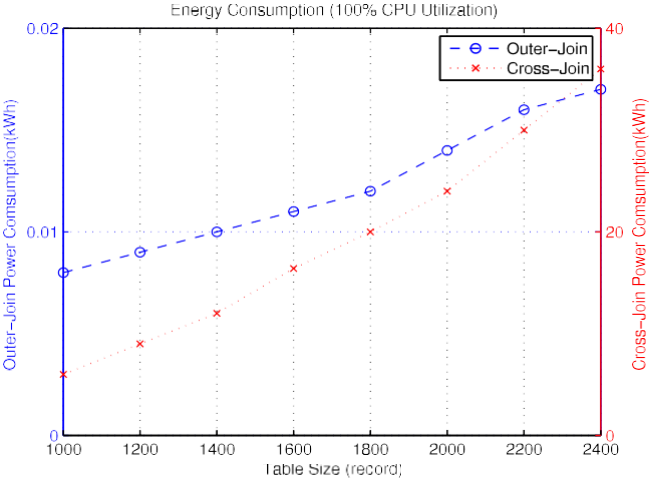
(a) Power consumption comparison between Outer-join and Cross-join in the 25% CPU utilization case.



(b) Power consumption comparison between Outer-join and Cross-join in the 50% CPU utilization case.



(c) Power consumption comparison between Outer-join and Cross-join in the 75% CPU utilization case.



(d) Power consumption comparison between Outer-join and Cross-join in the 100% CPU utilization case.

Fig. 10. Energy efficiency impact of outer-join and cross-join operations on multicore systems under various CPU utilization.

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. 9(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. 9(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 joinoperations.

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

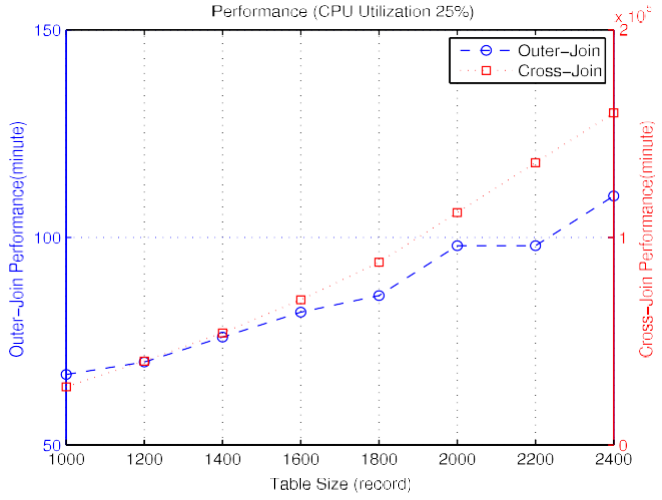
Now we compare the difference between the outer-join and cross join queries from the perspectives of CPU utilization impact on energy consumption. Again, we increase the table size from 1000 to 2400 records with an increment of 200. Fig. 10 reveals that regardless of CPU utilization, the cross-join query is a whole lot more energy expensive than the outer-join one. This trend is consistent with the results plotted in Fig. 9.

An intriguing observation drawn from Fig. 10 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 1000 to 2400. 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.

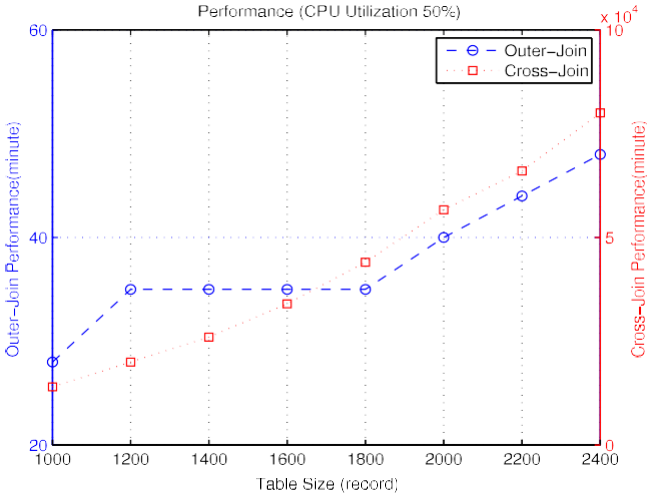
The implication of the results shown in Fig. 10 is that under heavy CPU utilization, reducing table size becomes a feasible approach to noticeably conserving energy consumption of the cross-join operation.

6.2.3. Energy efficiency vs. performance impacts

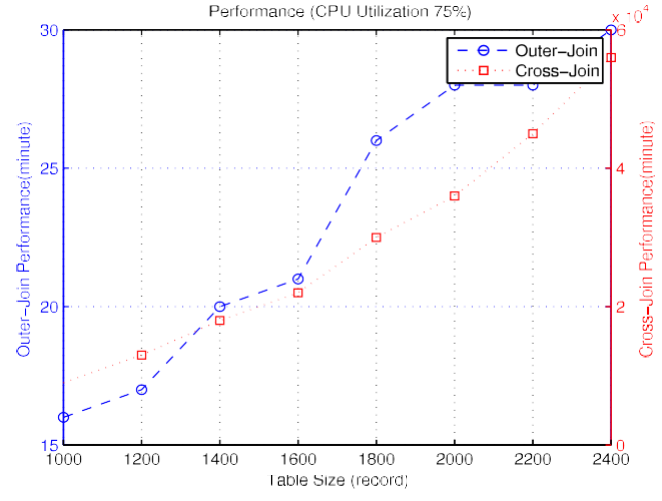
In this group of experiments, we compare the performance in terms of response time between outer-join and cross-join queries



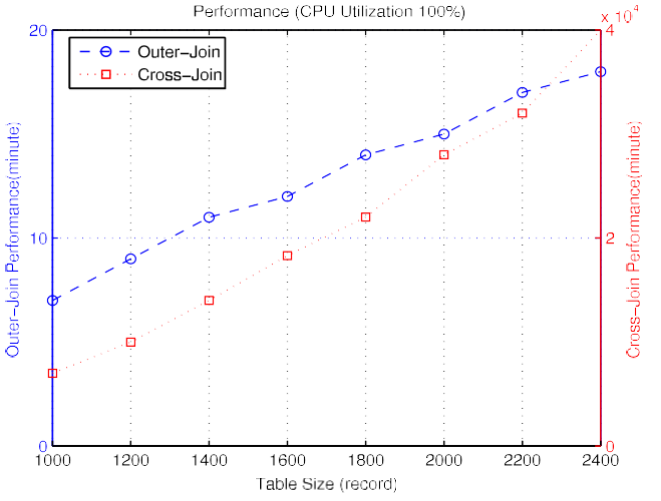
(a) Response time comparison between Outer-join and Cross-join queries in the multicore system when CPU utilization is 25%.



(b) Response time comparison between Outer-join and Cross-join queries in the multicore system when CPU utilization is 50%.



(c) Response time comparison between Outer-join and Cross-join queries in the multicore system when CPU utilization is 75%.



(d) Response time comparison between Outer-join and Cross-join queries in the multicore system when CPU utilization is 100%.

Fig. 11. Performance comparison of outer-join and cross-join queries in the multicore system under various CPU utilization.

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 6.2.1 and 6.2.2).

Like the configuration of the previous experiment, the table size is increased from 1000 to 2400 records with an increment of 200; the CPU utilization is set to 25%, 50%, 75%, and 100%, respectively. Fig. 11 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 1600 records and 75%, respectively. Not surprisingly, the performance trends revealed in Fig. 11 are similar to the energy-efficiency trends observed in Fig. 11. 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.

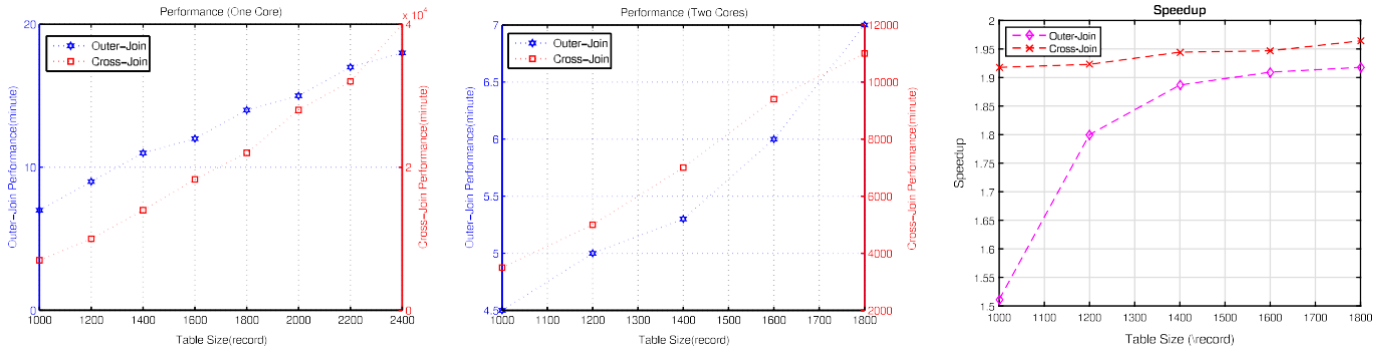
The results from this group of experiments also confirm that under high CPU workload (see, for example, Fig. 11(d)), reducing table size can significantly shorten the response times of the queries. This conclusion is especially true for the outer-join operation.

Fig. 12 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. 13 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. 13(a)) to two (see Fig. 13(b)). For instance, Fig. 13(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.

6.3. The indexing technique

Now we investigate the indexing technique's impacts on the energy behaviors of the outer-join queries. We only demonstrate

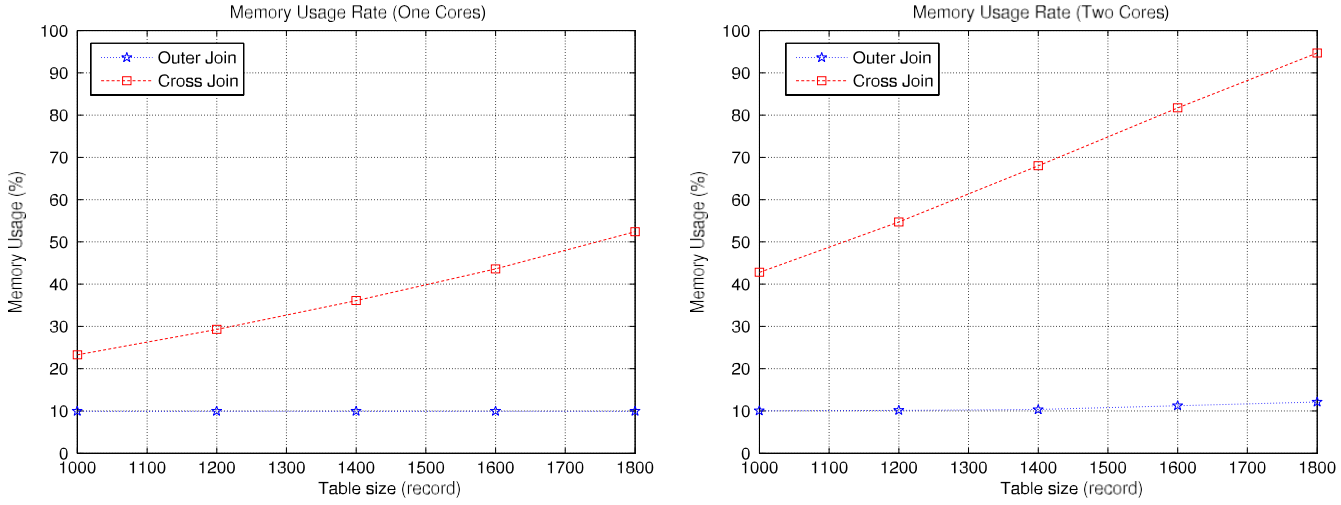


(a) Execution time comparison between Outer-Join and Cross-Join queries in single-core case.

(b) Execution time comparison between Outer-Join and Cross-Join queries in two-core case.

(c) Speedup comparison between Outer-Join and Cross-Join queries.

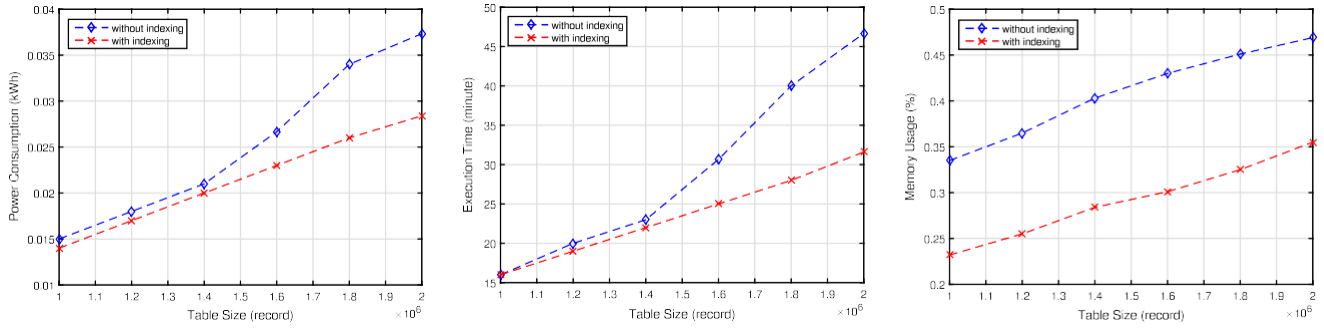
Fig. 12. Comparison of execution time between outer-join and cross-join under multicore situations.



(a) Memory usage comparison between Outer-Join and Cross-Join queries in single-core case.

(b) Memory usage comparison between Outer-Join and Cross-Join queries in two-core case.

Fig. 13. Impacts of outer-join and cross-join operations on the memory usage of multicore systems.



(a) Power consumption comparison between Cross-Join with indexing and Cross-Join without indexing.

(b) Execution time comparison between Cross-Join with indexing and Cross-Join without indexing.

(c) Memory usage comparison between Cross-Join with indexing and Cross-Join without indexing.

Fig. 14. Impacts of the indexing technique on energy efficiency of the outer-join and cross-join operations running in multicore systems.

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. 14 intuitively shows that indexing substantially affects the outer-join operation's power consumption, performance, and memory usage. The energy trend plotted in Fig. 14(a) is similar to the performance trend illustrated in Fig. 14(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.

Fig. 14(a) and (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. 14(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.

7. Conclusions and future work

We started this study 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.

We will pursue our future research direction into two steps. First, we plan to develop an energy efficient model aiming to predict energy-efficiency in database queries. In the second step, we plan to integrate the energy efficient model into an database system to optimize the energy efficiency of database operations. We also intend to seamlessly integrate the energy efficient model with a power manager and a thermal manager in the multicore-based database system to offer further energy savings.

References

[1] S. Tu, W. Zheng, E. Kohler, B. Liskov, S. Madden, Speedy transactions in multicore in-memory databases, in: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ACM, 2013, pp. 18–32.

[2] M. Lin, A. Wierman, L.L. Andrew, E. Thereska, Dynamic right-sizing for power-proportional data centers, IEEE/ACM Trans. Netw. 21 (5) (2013) 1378–1391.

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

[4] M. Ghamkhari, H. Mohsenian-Rad, Energy and performance management of green data centers: A profit maximization approach, IEEE Trans. Smart Grid 4 (2) (2013) 1017–1025.

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

[6] S.D. Viglas, A comparative study of implementation techniques for query processing in multicore systems, IEEE Trans. Knowl. Data Eng. 26 (1) (2014) 3–15.

[7] W. Lang, S. Harizopoulos, J.M. Patel, M.A. Shah, D. Tsirogiannis, Towards energy-efficient database cluster design, Proc. VLDB Endow. 5 (11) (2012) 1684–1695.

[8] S. Rivoire, M.A. Shah, P. Ranganathan, C. Kozyrakis, Joulesort: a balanced energy-efficiency benchmark, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, ACM, 2007, pp. 365–376.

[9] A.T. Clements, M.F. Kaashoek, N. Zeldovich, R.T. Morris, E. Kohler, The scalable commutativity rule: Designing scalable software for multicore processors, ACM Trans. Comput. Syst. (TOCS) 32 (4) (2015) 10.

[10] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.

[11] D. Tsirogiannis, S. Harizopoulos, M.A. Shah, Analyzing the energy efficiency of a database server, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ACM, 2010, pp. 231–242.

[12] D. Schall, T. Härder, Approximating an energy-proportional dbms by a dynamic cluster of nodes, in: Database Systems for Advanced Applications, Springer, 2014, pp. 297–311.

[13] Z. Xu, Y.-C. Tu, X. Wang, Pet: reducing database energy cost via query optimization, Proc. VLDB Endow. 5 (12) (2012) 1954–1957.

[14] S.-W. Ham, M.-H. Kim, B.-N. Choi, J.-W. Jeong, Simplified server model to simulate data center cooling energy consumption, Energy Build. 86 (2015) 328–339.

[15] C.M. Stoppel, F. Leite, Integrating probabilistic methods for describing occupant presence with building energy simulation models, Energy Build. 68 (2014) 99–107.

[16] Z. Xu, Y.-C. Tu, X. Wang, Dynamic energy estimation of query plans in database systems, in: 2013 IEEE 33rd International Conference on Distributed Computing Systems, (ICDCS), IEEE, 2013, pp. 83–92.

[17] A.-C. Orgerie, M.D.d. Assuncao, L. Lefevre, A survey on techniques for improving the energy efficiency of large-scale distributed systems, ACM Comput. Surv. (CSUR) 46 (4) (2014) 47.

[18] H. Ltaief, P. Luszczek, J. Dongarra, Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency, Comput. Sci.-Res. Dev. 27 (4) (2012) 277–287.

[19] D. Kliazovich, P. Bouvry, S.U. Khan, Greencloud: a packet-level simulator of energy-aware cloud computing data centers, J. Supercomput. 62 (3) (2012) 1263–1283.

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

[21] 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 Trans. Comput. Syst. (TOCS) 31 (3) (2013) 8.

[22] W. Zheng, S. Tu, E. Kohler, B. Liskov, Fast databases with fast durability and recovery through multicore parallelism, in: 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI 14, 2014, pp. 465–477.

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

[24] T. Endo, G. Jin, Software technologies coping with memory hierarchy of gpgpu clusters for stencil computations, in: 2014 IEEE International Conference on Cluster Computing, (CLUSTER), IEEE, 2014, pp. 132–139.

[25] F. Urbano, H. Dettki, Storing tracking data in an advanced database platform (postgresql), in: Spatial Database for GPS Wildlife Tracking Data, Springer, 2014, pp. 9–24.

[26] 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, Jun. 17 2014, uS Patent 8,756, 441.

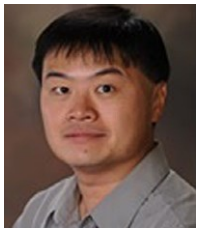
[27] F. Färber, S.K. Cha, J. Primsch, C. Bornhövd, S. Sigg, W. Lehner, Sap hana database: data management for modern business applications, ACM SIGMOD Rec. 40 (4) (2012) 45–51.

[28] G. Graefe, H. Volos, H. Kimura, H. Kuno, J. Tucek, M. Lillibridge, A. Veitch, In-memory performance for big data, Proc. VLDB Endow. 8 (1) (2014) 37–48.

Yi Zhou (S'16) received the B.E.E., M.S.E.E. degrees in electronic engineering all from Beijing University of Technology, Beijing, in 2006 and 2010 respectively. He is currently a doctoral student in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2013, he has been a software engineer in Alcatel-Lucent Technologies (China) Co.,Ltd. for four years. His research interests include energy-saving techniques, database systems, and parallel computing.

Shubbhi Taneja (S'15) received her B.E. degree from Maharishi Dayanand University, Haryana, India in 2012. She joined Samuel Ginn College of Engineering, Auburn University, to pursue as a doctoral student of Computer Science in 2013. Her main area of research include parallel and distributed systems, storage systems, energy-efficient computing, and performance evaluation. Her other areas of interest include web development and STEM programs for K-12 students. She is a student member of IEEE, ACM, and IEEE Computer Society.

Xiao Qin (S'00-'04M-SM'09) received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the Ph.D. degree in computer science from the University of Nebraska-Lincoln in 2004. He is currently a professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He won an NSF CAREER award in 2009. His research is supported by the US National Science Foundation (NSF), Auburn University, and Intel Corporation. He has been on the program committees of various international conferences, including IEEE Cluster, IEEE MSST, IEEE CCGrid, IEEE IPCCC, and ICPP. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He is a member of the ACM and a senior member of the IEEE.



Wei-Shinn Ku received his Ph.D. degree in computer science from the University of Southern California (USC) in 2007. He also obtained both the M.S. degree in computer science and the M.S. degree in Electrical Engineering from USC in 2003 and 2006, respectively. He is an Associate Professor with the Department of Computer Science and Software Engineering at Auburn University. His research interests include data management systems, data analytics, geographic information systems, mobile computing, and security. He has published more than 80 research papers in refereed international journals and conference proceedings. He is a senior member of the IEEE and a member of the ACM.

Jifu Zhang received the BS and MS in Computer Science and Technology from Hefei University of Technology, China, in 1983 and 1989, respectively. He received the Ph.D. degree in Pattern Recognition and Intelligence Systems from Beijing Institute of Technology in 2005. He is currently a Professor in the School of Computer Science and Technology at TYUST. His research interests include data mining and artificial intelligence, parallel and distributed systems.