Towards Energy-Efficient Multicore Database Systems

Yi Zhou*, Mohammed Alghamdi†, Shubbhi Taneja‡*, Wei-Shinn Ku*, Xiao Qin*
*Department of Computer Science and Software Engineering
Auburn University, Auburn, Alabama 36849-5347
Email: {yzz0074, shubbhi, wzk0004, xqin}@auburn.edu ‡corresponding author:shubbhi@auburn.edu
†Computer Science Department
Al-Baha University, Al-Baha City, Kingdom of Saudi Arabia

***Abstract*—In this paper, 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. 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). We make use of the proposed benchmark toolkit to quantitatively evaluate the performance of our novel multicore manager. Our 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.**

***Keywords*-Energy efficiency, database operations, multicore processors, benchmarks, data centers, database applications**

I. Introduction

High energy efficiency is of importance for reducing operating cost of data centers, where database applications are running on multicore servers [1]. 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 mulitcore 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.

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.

978-1-5090-5117-5/16/$31.00 ©2016 IEEE

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.

II. Related Work

Energy-efficient clusters are becoming increasingly popular in large-scale data centers [2][3] 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. 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 [4]. Lang *et al.* analyzed a number of important parameters related to the design of energy-efficient DBMS [5]. A framework built by Lang *et al.* optimizes queries by considering both performance and energy consumption as optimization criteria [5].

Although energy efficiency 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 [4]. Schall and Härder developed a distributed DBMS - WattDB - to make dynamical configurations to satisfy performance demands while conserving energy consumption on clusters [6]. Xu *et al.* implemented PET - an energy-aware query optimization framework that is an extension of the PostgreSQL kernel [7]. 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.

Energy consumption models have been developed for computing systems [8]. 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 [9]. Different from the existing energy consumption models, our models built in EDOM pay attention to modeling energy consumption of DBMS operations.

There is lack of research on modeling on the basic DBMS operations energy-efficiency. Our aim is investigate this area and give a guideline for further research.

III. Energy Efficiency Benchmark

In this section, we present the design issues of the energy efficiency benchmark toolkit for database systems. By dividing the toolkit into several modules, we obtain higher maintainability and extensibility.

A. Configuration Module

The responsibility of the workload generator is threefold. 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.

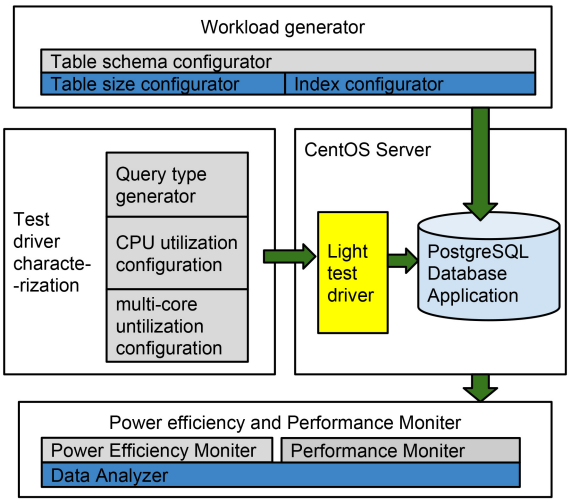


Figure 1: The framework of the energy efficiency benchmarking toolkit.

B. 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.

C. 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 [10]. 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 PostgreSQL, because the focus of our experiments is to measure energy consumption of database operations. Query requests are issued by the

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.

light-weight test driver, the energy consumption of which is ignored in our experiments. Table I shows the database server specification.

Table I 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

D. Power-Performance Monitor

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 PostgresSQL database server. The power meter employed in this study is TS-836A Plug Energy Watt Voltage Amps Meter (see Table II 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 II 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

IV. The Multicore Manager

The goal of the multicore manager is to make a good tradeoff between energy efficiency and performance in database systems.

A. Memory Usage Estimator

In modern database systems, memory resources become a vital component affecting performance and energy efficiency [11]. 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 [12].

Our empirical study reveals that the memory usage imposes a significant impact on the energy efficiency of database systems. For example, Figs. 2, 4(a), and 5(a) indicate that heavily utilized main memory adversely slowdowns 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

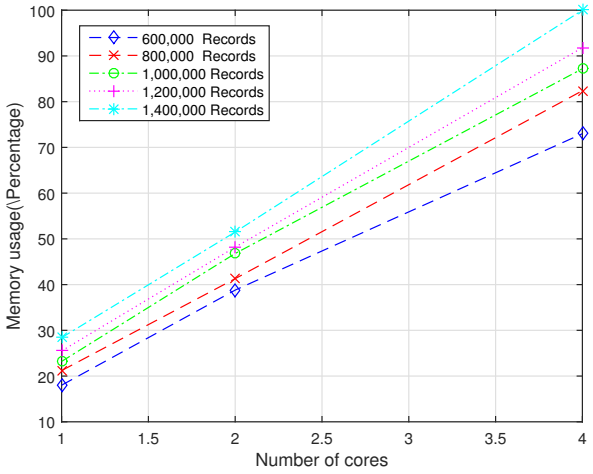


Figure 2: Memory load.

high level. Please refer to Section V-A for an 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 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 four operation factors, namely, database query type, the number of tables, the number of records, and the number of record size.

Given the numbers of tables and records in the table coupled with record size, the memory usage estimator applies a model to project memory usage, which is used in the multicore calculator to determine the number of the mulitcores.

B. 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 IV-A) 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

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.

problem.

Algorithm 1 Multicore Manager Algorithm: optimal()

Require:
number of tables t
number of records r
number of 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
number of 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 (Average case performance O(log n)). 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 IV-C) 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.

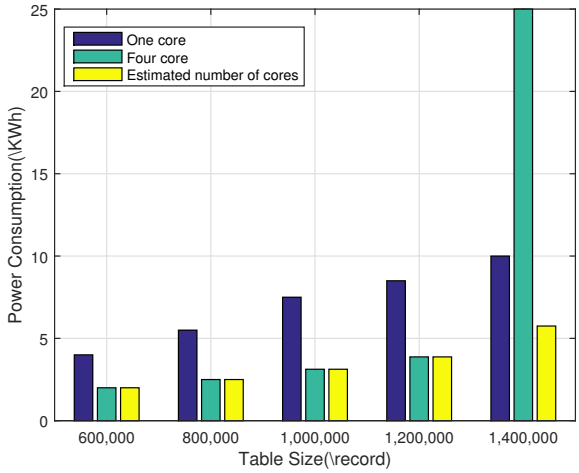


Figure 3: The validation of our multicore manager.

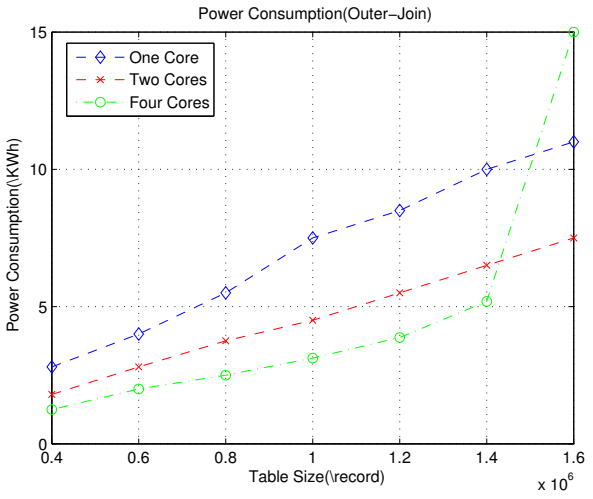
C. Energy Efficiency of the Multicore Manager

We design a group of experiments to study the energy efficiency of the multicore manager. We increase the table size from 6×10^5 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.

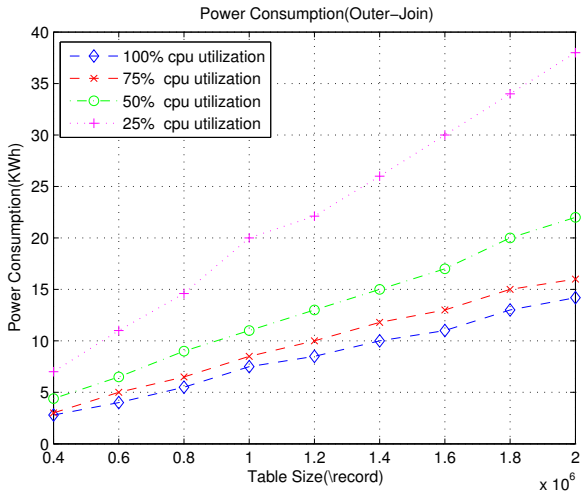
Fig. 3 shows the energy consumption of the multicore-based database system governed by our multicore manager; Fig. 3 also illustrates the energy consumption in the one-core and four-core cases. The results indicate that as the table size is increased from 6×10^5 to 1.2×10^6 records, the optimal number 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

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.

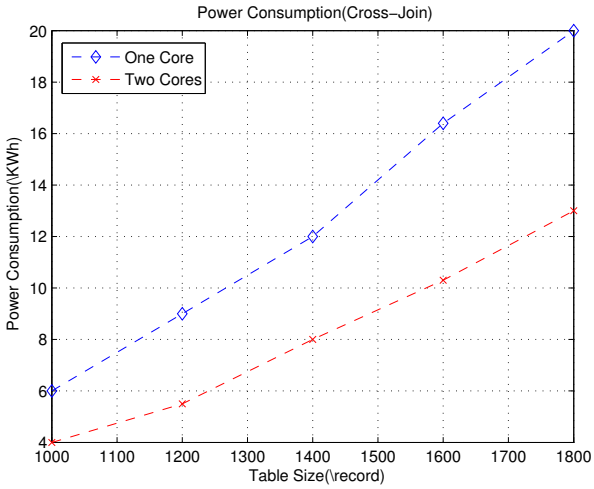


(a) Power Consumption with Multicores

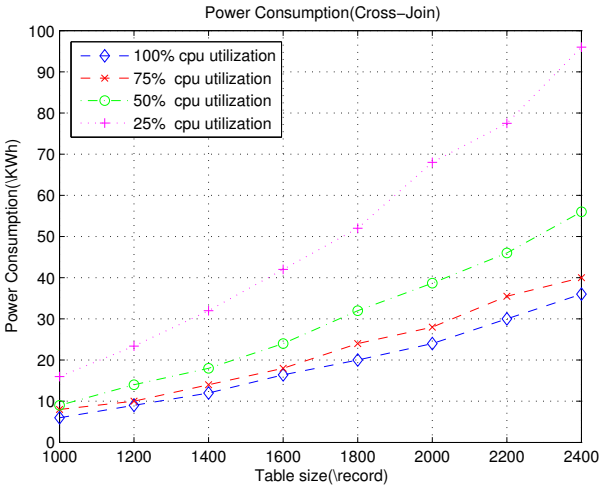


(b) Power Consumption under different CPU utilization

Figure 4: Power consumption profiling of query: Outer-Join



(a) Power Consumption with Multicores



(b) Power Consumption under different CPU utilization

Figure 5: Power consumption profiling of query: Cross-Join

Fig. 3). Our multicore manager judiciously downgrades the number of cores to three, which significantly conserves the energy consumption compared with the four-core case. The experimental result confirms that the multicore manager is capable of determining an optimal number of cores under relatively heavy workload.

V. Experimental Results

We have conducted extensive experiments to demonstrate the usage and effectiveness of EDOM. We focus on the impacts of CPU utilization and multicores on the energy efficiency and performance of the tested database system.

A. *Energy Efficiency of the Outer-Join Operation*

We issue a fixed number of outer-join queries while changing the database table sizes and number of cores in the system. Fig. 4(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. 4(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 pro-

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.

cessing 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. 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. 4(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 2 million is even 37% of the energy consumption of 25% CPU utilization. The reason is that the less CPU utilization used, the more idle status consumption is taken into account of the overall 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. Fig. 4(b) shows the database system's energy consumption of processing outer-join queries when CPU utilization is kept to 25%, 50%, 75%, 100%, respectively. 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%. 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.

B. Energy Efficiency of the Cross-Join Operation

Now we evaluate the energy efficiency of the database system using cross-join queries. We observe that various types of database operations have different impacts on energy efficiency. Nevertheless, the power consumption trend shown in Fig. 5(a) is similar to that of Fig. 4(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. 5(a) reveals that in the case where the table size is set to 1800, an extra core dramatically reduces the energy consumption. This result is consistent with the one drawn from Fig. 4(a).

The energy consumption of the three-core and four-core cases are not plotted in Fig. 5(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 out-join operations, the energy behavior of cross-join queries are more sensitive to main memory capacity of the system.

Fig. 5(b) confirms a trend illustrated in Fig. 5(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.

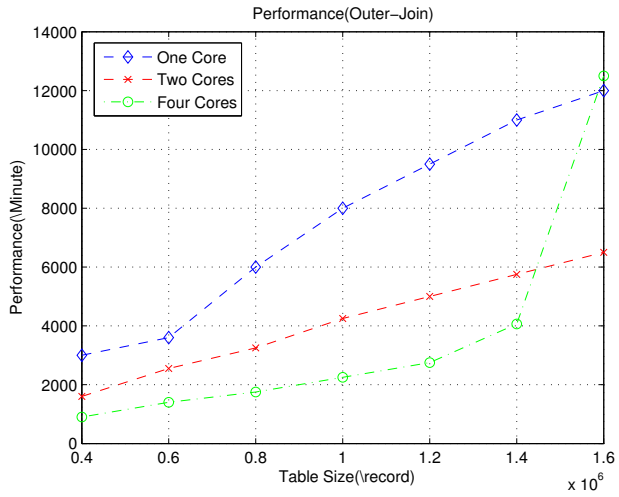
C. Performance of the Outer-Join Operation

Now we study performance of the outer-join operations. Fig. 6(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. 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.

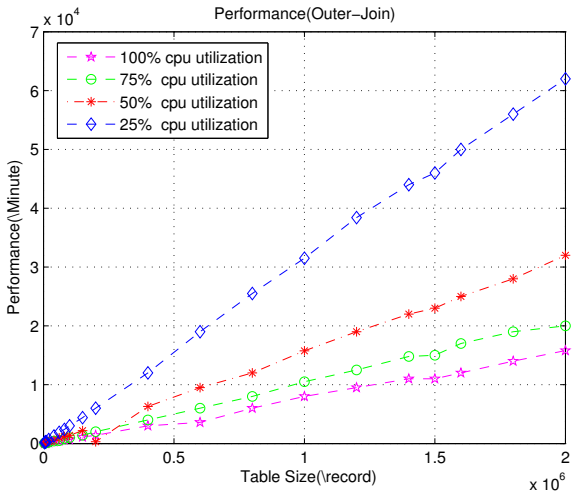
Fig. 6(b) illustrates the impact of CPU utilization on the performance of the system running outer-join operations. We vary the table size from 0.02 to 2.0 million; we also tested four cases where the CPU utilization is kept at 100%, 75%, 50%, and 25%, respectively. Fig. 6(b) shows that increasing CPU utilization shortens the time spent in performing the outer-join operations. 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(a) indicates that regardless of the number of cores, increasing the table size slightly drives the

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.

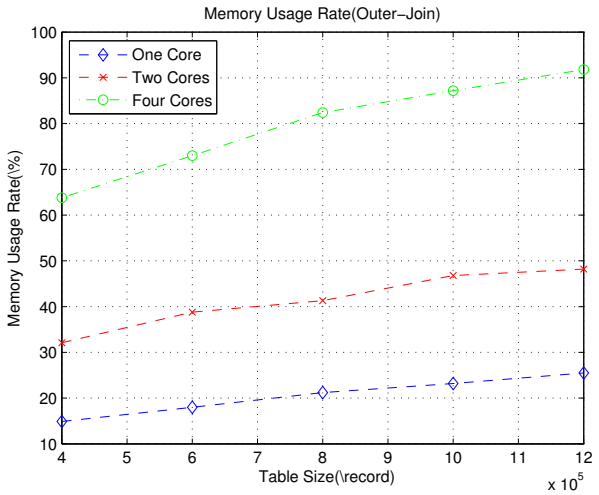


(a) Time Consumption with Multicores

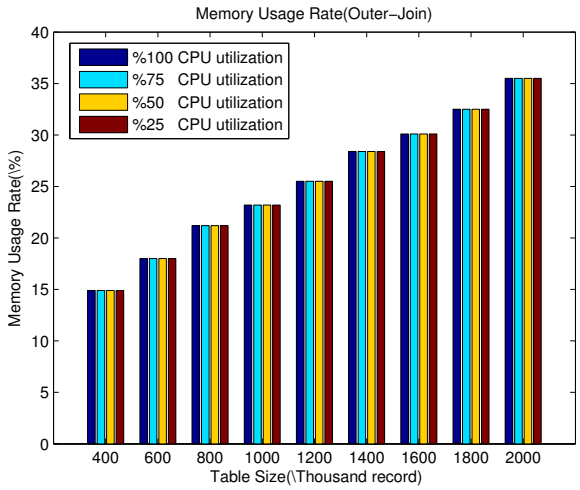


(b) Time Consumption under different CPU utilization

Figure 6: 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 7: Memory usage of Outer-Join under different CPU utilization and multicores.

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.

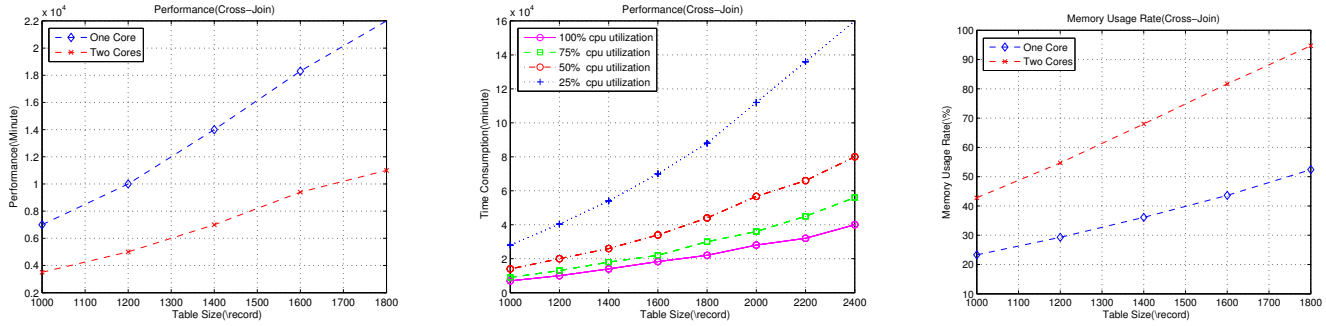
Fig. 7(b) reveals the impact of CPU utilization on system memory usage. The experimental results suggest that the CPU utilization has no noticeable impact on memory usage.

D. Performance of the Cross-Join Operation

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

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.



(a) Time Consumption with Multicores

(b) Time Consumption under different CPU utilization

(c) Memory Usage under different CPU utilization

Figure 8: Performance profiling of Cross-Join under different CPU utilization and multicores

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.

VI. Conclusion

In this study, we proposed *EDOM* - a tool systematically evaluating and optimizing the energy-efficiency of multicore-based 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.

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.

References

[1] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.

[2] 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*, vol. 60, no. 3, pp. 360–374, 2011.

[3] X. Ruan, X. Qin, Z. Zong, K. Bellam, and M. Nijim, "An energy-efficient scheduling algorithm using dynamic voltage scaling for parallel applications on clusters," in *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*. IEEE, 2007, pp. 735–740.

[4] 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*. ACM, 2010, pp. 231–242.

[5] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis, "Towards energy-efficient database cluster design," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1684–1695, 2012.

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

[7] Z. Xu, Y.-C. Tu, and X. Wang, "Pet: reducing database energy cost via query optimization," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1954–1957, 2012.

[8] 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*. IEEE Press, 2012, pp. 45–50.

[9] 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.

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

[11] 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*, vol. 40, no. 4, pp. 45–51, 2012.

[12] R. Karedla, J. S. Love, and B. G. Wherry, "Caching strategies to improve disk system performance," *Computer*, vol. 27, no. 3, pp. 38–46, 1994.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:03:39 UTC from IEEE Xplore. Restrictions apply.