GreenDB: Energy-Efficient Prefetching and Caching in Database Clusters

Yi Zhou, Shubbhi Taneja, *Member, IEEE*, Chaowei Zhang, and Xiao Qin, *Senior Member, IEEE*

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

Index Terms—Energy efficiency, prefetching, energy conservation

1 Introduction

Improving energy efficiency of parallel database systems becomes indispensable for building data centers supporting transnational database applications [1]. Traditional energy-saving techniques for clusters are inadequate for parallel database systems running on clusters. To address this problem, we focus on reducing energy consumption cost of large-scale database systems, which are comprised of multiple nodes or severs. We show how to develop a parallel database system called *GreenDB* - an energy efficient system running on clusters. At the heart of *GreenDB* is a prefetching and caching mechanism, which fetches hot data from passive nodes into active nodes. *GreenDB* offers energy savings by turning a large number of passive nodes into the low-power mode, which keeping a small number of active nodes to serve queries. The power management of *GreenDB* is governed by two mathematical models, namely, a prefetching model and an energy consumption model. We show how to build these two models and how to integrate the models into the prefetching and caching mechanism.

The following four motivations make energy-efficient prefetching and caching desirable and achievable.

- *Y. Zhou is with TSYS School of Computer Science, Columbus State University, 4225 University Avenue, Columbus, GA 31907-5645. E-mail: yzhou056@gmail.com.*
- *S. Taneja is with Department of Computer Science, Sonoma State University, 1801 E Cotati Ave, Rohnert Park, CA 94928. E-mail: taneja@sonoma.edu.*
- *C. Zhang and X. Qin are with the Department of Computer Science and Software Engineering, Auburn University, AL 36849-5347. E-mail: czz0032@tigermail.auburn.edu, xqin@auburn.edu.*

Manuscript received 6 June 2017; revised 24 Aug. 2018; accepted 27 Sept. 2018. Date of publication 4 Oct. 2018; date of current version 10 Apr. 2019. (Corresponding author: Shubbhi Taneja.)
Recommended for acceptance by D. Arnold.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2874014

1) Database clusters are cost-effective platforms for parallel database systems.
2) A pressing demand of reducing energy cost of large database systems.
3) Energy-saving techniques designed for single-server database systems are inadequate for DBMS clusters.

Database clusters have become a cost-effective computing platform to manage a massive amount of data for database management systems or DBMS [2], [3], [4]. Clusters achieve high-performance and low response time for database applications through parallel query processing on multiple database servers [5]. Data in clusters are partitioned and replicated among database nodes to handle queries in parallel [6].

Database Clusters - cost-efficient platforms - consist of multiple DBMS nodes being independent of one another. It is not uncommon for database clusters to employ a middleware layer to coordinate parallel queries evenly distributed among database nodes [7]. Such a middleware layer can easily and efficiently fetch and migrate data managed in the database nodes, because indexing large tables facilitates data migrations when data are replicated across multiple database nodes [8].

Energy cost is one of the significant components of operational costs in data center environments [9]. Evidence shows that a data center containing 1000 racks consumes 10MW total power per year [10]. 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 [11]. Unfortunately, we observe that little attention has been paid to energy efficiency improvement of database systems running on clusters. This observation motivates us to focus on energy-efficient database systems for modern data centers.

An increasing number of energy conservation techniques are adopted to build energy-efficient DBMS. For example,

1045-9219 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

power management [12], energy-efficient memory management [13], and energy-efficient virtual machines [14] can be applied to reduce energy consumption of database systems. These techniques become inadequate for database clusters, because the above energy-saving schemes improve DBMS energy efficiency without considering queries and data access patterns across multiple database nodes of a cluster.

A handful energy-saving techniques were designed for single-server database systems. For example, Chehadeh et al. investigated energy-efficient indexing strategies in an object-oriented database environment [15]. Poess and Nambiar proposed a way of making tradeoffs between existing power-saving techniques and their performance impact on database applications. Our analysis will guide system developers in making informed decisions regarding adopting power-preserving techniques. The aforementioned energy-saving schemes developed for single-server database systems are inadequate for DBMS on clusters. Moreover, energy-efficient prefetching and caching have not been incorporated in the prior database systems. We address this challenge by proposing an energy-efficient database cluster called *GreenDB*, in which prefetching and caching are seamlessly integrated to offer energy savings in a holistic way.

In the following example, we demonstrate that energy consumption of a DBMS cluster can be significantly conserved by duplicating popular tables housed on hot nodes while forcing cold nodes into the low-power mode.

Example 1. We setup a database cluster where there is no power-saving technique. Cold nodes in the low-power mode consume as low as 1.1 Watt, whereas the power of active nodes is 58 Watt. We employ the Wake-on-LAN [16] technique to wake up cold nodes, where the average power-transition overhead is approximately 297.5 Joules. Let us consider the following two strategies.

Strategy 1: No popular-table duplication; all database nodes are kept active.

Strategy 2: Duplicate popular tables from cold nodes into a hot node; turn of cold nodes to conserve energy; and keep the single hot node active.

In this example, we set tables size, inter-arrival delay, hit rate, and the number of cold nodes to 200K records, 16 seconds; 75 percent, and 2, respectively. The average power consumption of Strategy 1 and Strategy 2 are 417,005 and 189,737 joules, respectively. Strategy 2 to be deployed in GreenDB reduces energy consumption by 54.5 percent.

GreenDB strives to maintain popular tables on hot nodes being constantly active to server incoming queries. Due to the busy nature of hot nodes, idle time periods are less likely to occur on hot nodes. There is little incentive to place any hot node into the low-power mode. Thus, GreenDB aggressively keeps hot nodes active by adaptively duplicating popular tables and evicting non-popular ones. In contrast, cold nodes that exhibit excessively long idle time periods should be switched to the low-power state to improve energy efficiency. GreenDB creates ample opportunities to place cold nodes in the energy-efficient mode, thanks to a prefetching module (see Section 3.4) that fetches popular tables from cold nodes into hot ones.

An overarching goal of *GreenDB* is to place, access, and replicate data crossing multiple nodes on a database cluster in a way to minimize energy cost. Our *GreenDB* is comprised of multiple hot and cold database nodes (see Section 3, which execute queries in parallel. Hot nodes maintain popular data, the duplication of which can be found on cold nodes. When load is medium or low, *GreenDB* activates hot nodes to process queries while aggressively placing cold nodes into the low-power state to conserve energy. Under high workload conditions, *GreenDB* wakes up cold nodes to alleviate heavy load imposed on the hot nodes. To lower the overhead of node-level data management, *GreenDB* identifies popular data on cold node and makes replicates of the popular data on hot node for fast query process (see Section 3.3). *GreenDB* relies on prefetching and caching modules (see Section 3.4) to maintain data on hot and cold nodes in an energy-efficient way.

The rest of the paper is organized as follows. Section 2 summarizes the related work. The design and implementation of energy-efficiency *GreenDB* are discussed in Section 3. An energy-saving prediction model in GreenDB is presented in Section 4. Section 5 validates the energy efficiency of GreenDB by a set of experiments. Finally, Section 6 shows conclusions including the contributions of this research along with our future research directions.

2 Related Work

Clusters in modern data centers consume a large amount of energy in high-performance components [17]. A growing number of energy-saving techniques were proposed to improve the energy efficiency of data centers [18], [19]. For example, Nishikawa et al. proposed a power-saving method to improve the energy efficiency of multiple disk drives based on applications' I/O behaviors [20]. Ruan et al. developed an energy-efficient scheduling algorithm - TDVAS - using a popular dynamic voltage scaling technique to offer significant energy savings in severs [21]. Rather than investigating storage-level energy-efficient techniques, we focus on a holistic approach to the development of energy-efficient database clusters.

Prior studies revealed that energy-efficient database systems are affected by various factors. For example, Graefe's novel technique makes a server-class database management system energy efficient [22]; Graefe's findings advocate that I/O scheduling and placement offer opportunities for energy savings. Wang et al. recently investigated energy-saving data management techniques, indicating that data management software plays a vital role in boosting energy efficiency [23]. Theo et al. built a cluster framework, where individual nodes are dynamically attached and detached to a cluster on demand of dynamic workload [24].

Attention has been paid towards balancing energy efficiency and performance in the realm of database systems. Although energy savings and high performance are two conflicting design goals, our previous research [25] suggests that in a single database server, energy-efficient configurations may deliver good performance and high energy efficiency thanks to a significant reduction in idle time. We also showed that processors rather than memory and disks resources is a dominating factor in energy consumption.

Xu et al. implemented an energy-aware query optimization framework or PET as an extension of the PostgreSQL kernel [26]. PET makes use of its power cost estimation

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Comparisons between GreenDB and the Other Existing Schemes

Comparisons among the most closely related studies on energy-efficient database clusters				
Schemes	Data placement	Effective look-up	Energy efficiency	Platform design
AutoPlacer [29]	✓	✓	×	×
Self-Tuning TSX [30]	×	✓	×	×
BOSS [31]	×	✓	×	×
GreenDB	✓	✓	✓	✓

module and plan evaluation model to make good tradeoffs between energy efficiency and performance in database systems. Our *GreenDB* is distinctly different from the aforementioned approaches in that GreenDB conserves energy cost of distributed DBMS running on multiple servers while achieving high query performance through the data management architecture.

Very recently, researchers embarked on inspiring studies to model the energy consumption of database systems. Rivoire proposed approaches to modeling power consumption in components, systems, and data centers, aiming to improve energy-efficient system design [27]. A handful models were focused on profiling disk energy behaviors that depend on CPU demands [28]. Different from the existing energy consumption models, ours (i.e., the prefetching and power-saving models) built in GreenDB pay attention to modeling energy savings contributed by transferring popular tables from cold to hot nodes.

Emerging techniques have been proposed to optimize performance of database and in-memory storage systems. For example, Paiva et al.. proposed a self-tuning distributed NoSQL database system, which automatically optimize replica placement by leveraging locality patterns in data accesses [29]. Diegues et al. introduced a self-tuning approach that exploits lightweight reinforcement learning techniques to identify an optimal configuration to improve the performance of multi-core processors running parallel programs [30]. Xu et al. designed a framework that advocates blending on-demand and spot instances to slash cost for in-memory storage [31].

Table 1 summarizes the major technological discrepancies between GreenDB and the existing schemes reported in the literature. Let us compare GreenDB against the alternative solutions from the perspectives of data placement, effective look-up, energy efficiency, and platform design, respectively.

3 System Design

3.1 Basic Ideas

A database cluster governed by *GreenDB* is divided into a group of cold nodes and a group of hot nodes. The overarching goal is to keep the cold nodes in the low-power state (e.g., shutting down) for a long-time period, while making the hot nodes respond to queries accessing popular data tables. There are three motivations behind partitioning nodes into the hot and cold groups. Prior studies suggest that improving the utilization of an active database node leads to high energy efficiency [25], [32]. Second, placing an entire node into the low-power state is more energy efficient than turning off disks in the node. Third, reducing the number of power-state transitions can lower the power management overhead.

3.2 Software Architecture

We design the software architecture of *GreenDB* (see Fig. 1, which manages a master node and a group of hot and cold database nodes (or nodes for short). To achieve high performance of database clusters, GreenDB proactively maintains a group of hot nodes that are always in the active mode. Our preliminary findings suggest that high node utilization improves energy efficiency; CPUs are a dominate contributor to the energy consumption of database applications. Therefore, we skew load into a few active hot nodes to boost the hot-node utilization and energy efficiency.

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

GreenDB consists of four software modules, namely, the query manager (see Section 3.3), the data prefecthing module (see Section 3.4), the energy-cost model (see Section 3.5), table replication module, and power manager (see Section 3.6). Queries issued by database applications are managed by the query manager. The data prefecthing module is focused on fetching popular data tables from cold nodes into hot nodes. The energy-cost model offers an energy-saving guideline for the prefetching module. The table replication module creates duplicated tables on hot nodes according to prefetching decisions. The power manager in each database nodes is responsible to decide the power state of each node. To improve energy efficiency, the prefetching module makes use of an energy cost model to estimate energy overhead and impact of prefectching decisions. After a decision is made by the prefetching module, the

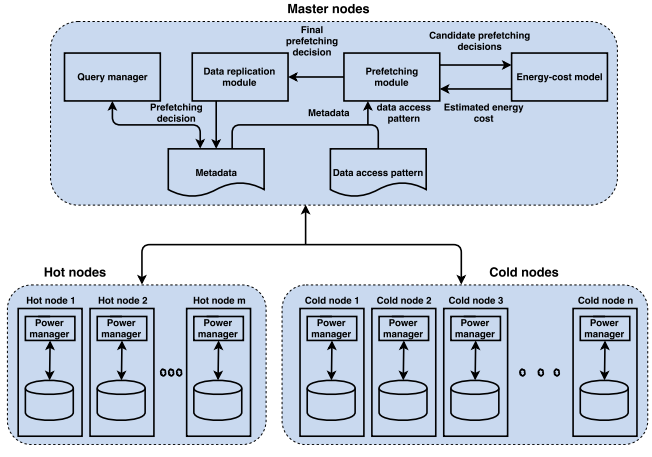


Fig. 1. The architecture of the GreenDB system.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

data replication module duplicates popular tables from cold into hot nodes in a batch manner.

The functionality of the table replication module is to create replicas of popular tables from on hot nodes. The replication module is governed by prefetching decisions yielded by the prefetching module (see Section 3.4). Each prefetching decision specifies three items, namely, a table to be duplicated, a source node, and a destination node. With the table replication and prefetching modules in place, duplicated tables stimulate hot nodes to perform as data caches tailored for cold nodes in the GreenDB.

3.3 Query Manager

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

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

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

3.4 Prefetching Module

The prefetching module periodically coordinates with the metadata manager and the access pattern manager to make prefectching decisions, which in turn are infused into the table replication module (see the last paragraph in Section 3.2).

The prefetching module has two salient features. First, the module keeps track of the access patterns of records in tables to pinpoint popular data. Second, the module creates replicated copies of popular tables to be placed on the most appropriate hot nodes. The decisions of making replicated tables largely depends on potential energy savings, which are predicted by the energy-cost model (see Section 3.5).

To make decisions on fetching data tables from cold nodes, the prefetching module assesses four situational factors including (1) the locations of candidate tables, (2) the popularity of data accessed by queries, (3) estimated energy savings, and (4) query predictions. It is noteworthy that the energy-cost model (see Section 3.5) estimates energy savings (i.e., factor 3) gained by fetching candidate tables. Query predictions (i.e., factor 4) are made possible by exploring historical query access patterns. We advocate plugging in query-prediction algorithms into GreenDB (see Fig. 1) to governs the process of prefetching tables. The development of query-prediction algorithms is out the scope of this study. To simplify the implication while focusing on the prefetching algorithm, we assume that query access patterns are given apriori.

The existing prefetching scheme in *PostgreSQL* relies on the *synchronous replication* module to maintain duplicated data between cold and hot nodes. Such a synchronous-based prefetching strategy (or *SyncPrefetch* for short) in PostgreSQL is expensive when it comes to maintaining replicated data on hot nodes. Before committing an update transaction, *SyncPrefetch* must obtain exclusive locks on all the copies. The transaction may have to send lock requests to cold nodes, waiting for the locks to be granted. During such a potentially long time period, *SyncPrefetch* continues holding all the other locks. If the cold nodes or communication links fail, the transaction cannot commit until all the nodes storing modified data are recovered the failure. Even if the locks are obtained readily without encountering any failure, committing a transaction has to deliver a few additional messages as specified by the commit protocol.

Unlike *SyncPrefetch*, the prefetching scheme in GreenDB takes the full advantage of asynchronous replications to cut energy cost of fetching hot data from cold nodes. Similar schemes have been applied in database systems (see, for example, [33]). GreenDB keeps newly updated data in hot nodes without aggressively pushing the data to the cold nodes. Such a lazy-update strategy achieves high energy efficiency by avoiding excessive power-state transitions occurred in the cold nodes. When additional popular data are in a process of being transferred from the cold to hot nodes, the unsynchronized data will be copied back to the cold nodes. GreenDB's the prefetching module harvests energy savings thanks to two reasons. First, the cold nodes are forced to stay in the low-power mode for an increased time period. Second, the number of power-state transitions in the cold nodes is substantially reduced.

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

The prefetching algorithm (see Algorithm 1) outlined above initializes the query and the number of cold nodes to q and m, respectively . Next, if the $table(q)$ is resident in the hot node, it server the query by hot node, otherwise it will power up the cold node in which the $table(q)$ is located and fetch the $table(q)$ into the hot node from the cold node(see Lines 1-7). Finally, we estimate the energy-efficiency of the tables in the lookahead table, fetch most energy-efficiency tables of whole GreenDB into the hot node if they are not kept in the hot node (see Lines 9-11). Unpopular tables are evicted from the hot nodes to release space for newly fetched ones (see Line 12).

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1. GreenDB Prefetching: $Prefetching()$

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

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

3.5 Energy-Cost Model

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

The energy-cost model integrates profiling results and the prefetching algorithm to offer us an interface to estimate energy savings provided by data tables in hot nodes. Profiling results take into account of three factors - energy consumed on fetching data from cold nodes, energy spent by hot nodes to store prefetched data tables, and energy overhead caused by power-state transitions. Our energy-cost model leverages the profiling data of energy consumption to estimate energy savings of offered by prefetched table. To build the energy-cost model through an energy profiling study, we independently connect hot nodes, cold nodes, and network interconnection to three power sockets; the displays are plunged into a separate power socket, ensuring that measured energy readings is contributed by individual devices being gauged. The power meter employed in this study is TS-836A Plug Energy Watt Voltage Amps Meter (see TABLE 2 for details).

3.6 Power Manager

The power manager in GreenDB dynamically and automatically changes the power mode of cold nodes to converse energy. For instance, if a cold node has been sitting idle for a given threshold, the power manager immediately transition the node into the low-power mode. Such a threshold can be automatically adjusted in GreenDB depending on workload conditions. For another example, prior to prefetching tables from a cold node to a hot node, the power manager wakes up the cold node.

A power-consumption monitor is built-in each of cold nodes to keep track of access activities in the cold nodes.

TABLE 2
Notation for the Description of the Prefetching Module

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

The monitor initiates power-transition requests to the corresponding power manager of a cold node. Upon the arrivals of power-down requests, the manager immediately powers down the cold nodes. Since each cold node has its individual access history, the power manager dynamically adjusts spin-down intervals according to the access activities to improve the energy efficiency of each cold node.

4 Energy-Efficient Prefetching

We develop in *GreenDB* an energy-saving prediction model, which governs the energy-saving calculation in *Steps* 9 and 10 in algorithm $Prefetching$. The prediction model along with the calculation module is indispensable for GreenDB's prefetching module, because the model speculates the amount of energy conserved by fetching candidate tables from cold nodes into hot nodes. The model also calculates potential cost of caching a hot-node-resident table rather than evicting the table from its hot node. Table 4 summarizes the notation used throughout this section.

To analyze circumstances under which prefetching database tables can yield energy savings, we focus on a single referenced table stored in a node. Let $Q_j \subseteq L$ be a set of queries accessing tables in the jth node. Thus, Q_j - a subset of lookahead L - can be defined as $Q_j = \{q|(q \in L) \cap (node(q) = j) \cap (\text{table}(q) \notin H).\}$, where $node(q) = j$ means query q is accessing the jth node; $\text{table}(q) \notin H$ indicates that the table accessed by query q has not been fetched into hot node denoted as H.

Given a set $Q_{kj} \subseteq Q_j$ of queries accessing the kth table τ_{kj} in the jth cold node, we derive the energy saving $E_{Saving}(\tau_{ij})$ achieved by fetching table τ_{kj} from the cold node into a hot node. Q_{kj} is comprised of all the queries referencing a common table τ_{kj} that is not present in the hot node; therefore, Q_{kj} can be formally expressed as $Q_{k,j} = \{q\big|(q \in Q_j) \cap (table(q) = \tau_{k,j}) \cap (\tau_{k,j} \notin H).\}$, where $table(q) = \tau_{k,j}$ means query q is accessing table $\tau_{k,j}$; $\tau_{k,j} \notin H$ suggests that table $\tau_{k,j}$ is not residing in the hot node. Intuitively, energy savings $E_{saving}(\tau_{kj})$ can be computed by considering the energy consumption incurred by each query in Q_{kj}.

4.1 Two Energy Saving Principles

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

Energy-Saving Principle 1. GreenDB aims not only to enlarge idle periods but also to increase the number of idle

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Power Meter Specifications

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

periods in database nodes. Idle periods should be larger than break-even time T_{BE} to offset the cost of entering the low-power state. Principle 1 can be realized by combining two adjacent idle periods to form a single idle period being larger than T_{BE}. GreenDB prefetches a table accessed by adjacent queries separated by idle periods, thereby forming a large idle time allowing the node to enter the low-power state to conserve energy.

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

4.2 Modeling Energy Savings

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

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

After table τ_{kj} is fetched to the hot node, cold node j's set $\Omega_j = \{\alpha_{1j}, t_{1j}, \alpha_{2j}, t_{2j}, ...\alpha_{ij}, t_{ij}, ...\alpha_{nj,j}, t_{nj,j}\}$ of access activities for queries in Q_j must be updated by deleting any interval $t_{ij} \in \Omega_j$ in which τ_{kj} is accessed.

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

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

TABLE 4
Notation for the Description of the Energy-Saving Calculation Module

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

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

$$E_{Saving}(\tau_{ij}) = E_{non\text{-}PM} - (E_{Cold} + E_{Hot}), \tag{1}$$

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

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

Case 1. Both the ith and $(i+1)$th idle periods are equal to or smaller than break-even time T_{BE}. Thus, we have $0 < \alpha_{ij} \leq T_{BE}, 0 < \alpha_{(i+1)j} \leq T_{BE}$, and $\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} > T_{BE}$. This condition implies that the jth cold node is in the idle mode during α_{ij} and $\alpha_{(i+1)j}$. Energy consumption experienced by the node in active period t_{ij} is $P_A \cdot t_{ij}$. Hence, energy consumption E_{non-PM} in case 1 can be expressed as:

$$E_{non\text{-}PM} = P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) + P_A \times t_{ij}. \tag{2}$$

When table τ_{ij} is prefetched, a large (i.e., larger than T_{BE}) idle period can be formed by combining periods t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$. Therefore, E_{Cold} can be computed as the energy consumption of the jth cold node in the standby mode during

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$. Taking into account energy overhead of power-state transitions, we can calculate energy consumption E_{Cold} of cold node using the equation below:

$$E_{Cold} = P_S \times (\alpha_{ij} + t_{ij} + \alpha_{(i+1)j} - T_D - T_U) + E_D + E_U, \quad (3)$$

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

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

$$E_{Hot} = P_A \times t_{ij}. \quad (4)$$

$E_{Saving}(\tau_{ij})$ in case 1 can be determined by substituting (2)-(4) into (1). Hence, we have:

$$\begin{aligned} E_{Saving}(\tau_{ij}) = {} & P_I \times (\alpha_{ij} + \alpha_{(i+1)j}) \\ & - P_S \times (\alpha_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) \\ & - E_D - E_U. \end{aligned} \quad (5)$$

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

The jth cold node in this case is transitioned into standby during $\alpha_{(i+1)j}$, since $\alpha_{(i+1)j}$ is larger than T_{BE}. The energy consumption of the cold node in $\alpha_{(i+1)j}$ is expressed as (see the third term on the right-hand side of (6) below). Thus, the energy consumption E_{non-PM} of cold node during t_{ij}, α_{ij}, and $\alpha_{(i+1)j}$ is

$$E_{non-PM} = P_I \times \alpha_{ij} + P_A \times t_{ij} + (P_S \times (\alpha_{(i+1)j} - T_D - T_U) + E_D + E_U). \quad (6)$$

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

$$E_{saving}(\tau_{ij}) = P_I \times \alpha_{ij} - P_S \times (\alpha_{ij} + \tau_{ij}). \quad (7)$$

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

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

$$E_{saving}(\tau_{ij}) = P_I \times \alpha_{(i+1)j} - P_S \times (\alpha_{(i+1)j} + T_{ij}). \quad (8)$$

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

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

$$\begin{aligned} E_{non-PM} = {} & P_A \times t_{ij} + \big(P_S \cdot (\alpha_{ij} - T_D - T_U) + E_D + E_U\big) \\ & + \big(P_S \times (\alpha_{(i+1)j} - T_D - T_U) + E_D + E_U\big), \end{aligned} \quad (9)$$

where the second and third term on the right hand side of (9) are the energy consumed by the cold node in standby periods α_{ij} and $\alpha_{(i+1)j}$, respectively. With a hot node in place, energy consumption E_{Cold} and E_{Hot} in this case are the same as in case 1 (see also (3) and (4)). Therefore, the energy saving $E_{Saving}(\tau_{ij})$, is derived from E_{non-PM} (see (9)), E_{Cold}, and E_{Hot} as

$$E_{Saving}(\tau_{ij}) = E_D + E_U - P_S \times (T_D + T_U + t_{ij}). \quad (10)$$

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

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

$$E_{Cold} = P_I \times (\alpha_{ij} + t_{ij} + \alpha_{(i+1)j}). \quad (11)$$

The values of E_{non-PM} and E_{Hot} are the same as those of case 1 (see also (2)). Applying E_{non-PM}, E_{Trans} and $E_{GreenDB}$ to (1), we estimate the negative energy-saving impact as:

$$E_{saving}(\tau_{ij}) = -P_I \times t_{ij}. \quad (12)$$

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

$$\Omega_{k,j} = \Omega_{k,j,1} \cup \Omega_{k,j,2} \cup \Omega_{k,j,3} \cup \Omega_{k,j,4} \cup \Omega_{k,j,5}, \quad (13)$$

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

1) for case 1:

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

2) for case 2:

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

3) for case 3:

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

4) for case 4:

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

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

5) for case 5:

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

Now we derive energy saving $E_{Saving}(\tau_{kj})$ offered by fetching table τ_{kj} from cold node j to the hot node. Thus, we obtain $E_{Saving}(\tau_{kj})$ from (5), (7), (8), (10), (11) and (12), where the last item on the right hand side is the energy overhead of fetching τ_{kj} from cold node j to the hot node.

$$\begin{aligned} E_{Saving}(\tau_{k,j}) &= \sum_{T_{ij}\in\Omega_{k,j}} (E_{Saving}(\tau_{ij})) \\ &= \sum_{T_{ij}\in\Omega_{k,j,1}} (P_I \cdot (\alpha_{ij} + \alpha_{(i+1)j}) \\ &\quad -P_S \cdot (\Omega_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U) \\ &\quad + \sum_{T_{ij}\in\Omega_{k,j,2}} \big(P_I \cdot \alpha_{ij} - P_S \cdot (\alpha_{ij} + t_{ij})\big) \\ &\quad + \sum_{T_{ij}\in\Omega_{k,j,3}} \big(P_I \cdot \alpha_{(i+1)j} - P_S \cdot (\alpha_{(i+1)j} + t_{ij})\big) \\ &\quad + \sum_{T_{ij}\in\Omega_{k,j,4}} \big(E_D + E_U - P_S \cdot (T_D + T_U + t_{ij}).\big) \\ &\quad - P_I \cdot \sum_{T_{ij}\in\Omega_{k,j,5}} t_{ij} - P_A \cdot \text{time}(\tau_{k,j}). \end{aligned} \tag{14}$$

4.3 Calculating Energy Savings

We show how to plugin the model (see Section 4.2) into an algorithm to calculate energy conserved by *GreenDB*. Given the kth table τ_{kj} residing in cold node j, Algorithm 2 computes energy savings of prefetching table τ_{kj} from the cold nodes to the hot node.

All of the energy-saving cases presented in Section 4.2 are explicitly handled from Steps 1 through 14 (See Algorithm in 2: Case 1 in Line 7, Case 2 in Line 9, Case 3 in Line 12, Case 4 in Line 15), whereas Step 18 addresses the issue of a negative impact on energy savings.

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

5 Experimental Results

In our extensive experiments, we leverage ten power meters (see Table 3) to precisely measure energy consumption of our testbed, which is a 20-node cluster (see Table 4). As a default setting where there are five nodes, our GreenDB orchestrates a single hot node coupled with four cold nodes (i.e., 5-node cluster). The sole purpose of this default configuration is to highlight GreenDBs strength in conserving energy consumption. More importantly, we manipulate the default configuration by vary the system parameters such as data size. For example, to investigate the scalability of GreenDB, we increase the number of cold nodes from four (i.e., a default value) to eight and twelve, respectively (see Section 5.4).

To quantitatively evaluate *GreenDB*, we first compare our prefetching strategy with the one implemented in Postgresql using the build-in stream replication scheme (see Section 5.2). We measure energy consumed by replicating one data table from a cold node to a hot node. The purpose of this set of experiments is to investigate an energy-efficient way of managing, replicating, and storing data in GreenDB. Creating data replications on hot nodes has a strong impact on energy consumption and; therefore, we shed light on the impacts of vital factors on the two evaluated prefetching strategies in GreenDB. The factors considered in our experiments include data size (see Section 5.3), the number of nodes (see Section 5.4), hit rate (see Section 5.5), inter-arrival times (see Section 5.6), and the number of disks in a cold node (see Section 5.7).

Algorithm 2. Calculating Energy Savings

Input:
table $\tau_{k,j}$
cold node j
a set Ω_j **of queries accessing cold** $node_j$
Output:
$E_{Saving}(\tau_{k,j})$
1: update $E_{Saving}(\tau_{k,j}) \leftarrow 0$;
2: **for** $j = 1$ to m **do**
3: **for** $i = 1$ to n_j **do**
4: **if** $alpha_{ij} + t_{i,j} + i_{(i+1)j} > T_{BE}$ **then**
5: **if** $0 < i_{i,j} \leq T_{BE}$ **then**
6: **if** $0 < i_{i,j} \leq T_{BE}$ **then**
7: **if** $0 < i_{(i+1)j} \leq T_{BE}$ **then**
8: $E_{saving}(\tau_{ij}) = P_I \times (\alpha_{ij} + \alpha_{(i+1)j}) - P_S \cdot (\alpha_{ij} + t_{ij} + \alpha_{(i+1)} - T_U - T_D) - E_D - E_U$
9: **else**
10: $E_{saving}(\tau_{ij}) = P_I \cdot \alpha_{ij} - P_S \cdot (\alpha_{ij} + \tau_{ij})$
11: **end if**
12: **else**
13: $E_{saving}(\tau_{ij}) = P_I \cdot \alpha_{(i+1)j} - P_S \cdot (\alpha_{(i+1)j} + T_{ij})$
14: **end if**
15: **else**
16: $E_{saving}(\tau_{ij}) = E_D + E_U - P_S \times (T_D + T_U + t_{ij})$
17: **end if**
18: **else**
19: $E_{saving}(\tau_{ij}) = -P_I \times t_{ij}$
20: **end if**
21: **end for**
22: **end for**
23: **return** $E_{saving}(\tau_{ij}) - P_A \times time(\tau_{kj})$

5.1 Experimental Setups

All the storage nodes managed by GreenDB are running the PostgreSQL database system powered by CentOS. PostgresSQL, a highly extensible object-relational database management system, securely stores and retrieves a variety of data. PostgreSQL is capable of processing workloads ranging from small scale applications to large Web-based applications. Table 5 summarizes the configurations of the database nodes, which resemble real-world servers that have been widely adopted in data-intensive environments. We create and maintain a dedicated computing environment to test our prefetching strategy implemented in GreenDB - a cluster database system. The focus of our empirical study is to explore and validate the energy efficiency of GreenDB on clusters. PostgresSQL facilitates a convenient way of managing and transferring replicated data on hot nodes in GreenDB. We compare our energy-

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

TABLE 5
Parameters and Configurations of Database Nodes (**OptiPlex 3020**)

Parameter	Value	Parameter	Default
Active Power: P_A	58 W	*Intervals*	15 Sec
Idle Power: P_I	27.7 W	*Num of Record*	200 K
Standby Power: P_S	2.5 W	*Hit Rate*	75%
Wake Up Energy: E_U	35 J	*Disk/Node*	2
Sleep Down Energy: E_D	32 J	*Num of Hot Node*	1
Wake Up Time: T_U	8.5 s	*Num of Cold Node*	4
Sleep Down Energy: T_D	3.3 s	*Break-even Time:* T_{BE}	13.3 s

TABLE 6
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

efficient prefetching scheme with the existing solution offered in PostgresSQL.

Table 6 shows the specifications of the cluster running the tested database system. We measure table sizes in terms of the number of records rather than physical data size, because the number of records is more intuitive than data volume. We compare GreenDBs power management (see also Section 3.6) with the conventional dynamic power management [20], where nodes are automatically transitioned into the low-power mode after sitting idle for a specified time (e.g., 14 seconds). For fair comparison purpose, we integrate the dynamic power management strategy with PostgreSQL.

5.2 Prefetching Data to Hot Nodes

The prefetching mechanism plays an important role in reducing energy consumption of database systems running on clusters. In this part of the study, we pay attention to the energy efficiency of the prefetching scheme that creates and transfers replicated data from cold nodes to hot ones in GreenDB. In particular, we evaluate the impact of the data table size on energy savings offered by our prefetching strategy. Recall that *SyncPrefetch* in PostgreSQL is expensive in terms of energy consumption (see Section 3.4). In this group of experiments, we show that the prefetching scheme in GreenDB employs asynchronous replications to minimize energy cost of fetching hot data from cold nodes.

For simplicity, we keep one hot node in GreenDB. Nevertheless, our findings indicate that the energy-efficiency trend of multiple-hot-node cases is similar to that of the single-hot-node case. The experimental results for multiple-hot-node scenarios can be found in Section 5.4.

5.2.1 GreenDB versus SyncPrefetch

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

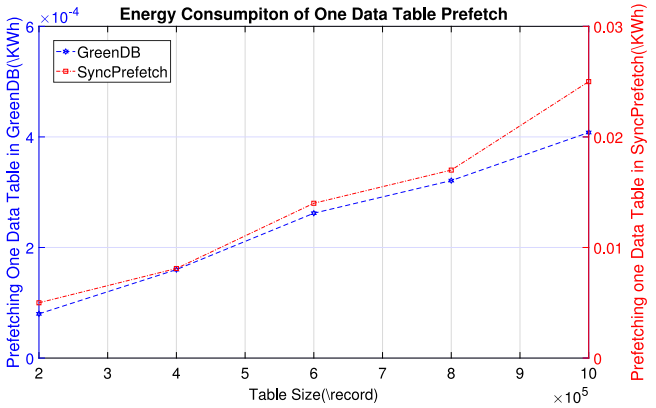


Fig. 2. Energy consumption of prefetching one data table from a cold node to a hot node. The actual table size is set to 368 MB, 736 MB, 1100 MB, 1500 MB, 1800 MB, respectively.

Fig. 2 shows the energy consumption of prefetching a data table in GreenDB and SyncPrefetch. The results depicted in Fig. 2 indicate that GreenDB is superior to SyncPrefetch in terms of energy efficiency. More specifically, GreenDB improves the energy efficiency of SyncPrefetch by a factor up to 60.9 with an average of 52.5. We also observe from Fig. 2 that the energy consumption of both prefetching strategies goes up as the fetched table size increases. In Fig. 2, the actual size in terms of number at each point of the *X*-axis is 368 MB, 736 MB, 1100 MB, 1500 MB, 1800 MB respectively, we make use of the unit in record size instead of physical size to show how the database table size will impact on the energy efficiency straightforwardly.

GreenDB substantially reduces energy consumption of prefetching data by replacing the synchronous replication module in SyncPrefetch with the asynchronous one (see the details in Section 3.4). SyncPrefetch offers slim opportunities for hardware resources (e.g., CPUs, hard drives) in cold nodes to transition into the low-power state, thereby leading to high energy consumption. In contrast, the prefetching strategy in our GreenDB fetches hot data in a batch manner, which keeps the hardware resources in the energy-saving mode for a long time period after the hot-data prefetching process is completed.

Fig. 3 shows the energy-saving ratio of GreenDB over SyncPrefetch. The results plotted in Fig. 3 indicate that GreenDB conserves the energy consumption of SyncPrefetch by up to 98.4 percent. The energy-saving ratio is extremely high when the data table size is either small (e.g., $2 * 10^5$records) or large (e.g., $1 * 10^6$records). We conclude that GreenDB is more energy efficient than PostgreSQL's build-in SyncPrefetch.

5.2.2 Energy Consumption and Execution Time

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

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

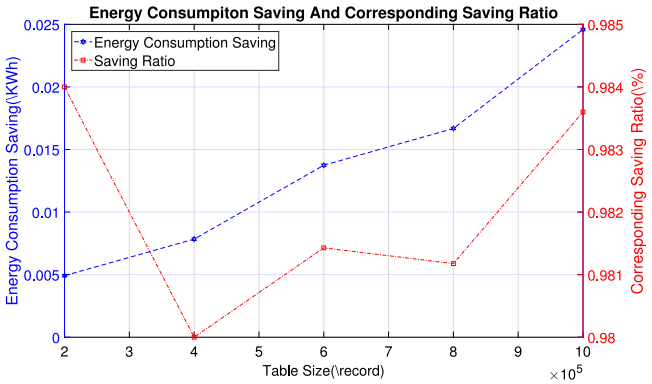


Fig. 3. Energy consumpiton saving and corresponding saving ratio.

Fig. 4c reveals that our GreenDB is faster than SyncPrefetch during the data transfer from cold to hot nodes. For example, it takes less than 30 seconds to create a replicated table of 2×10^5 records on the hot node, whereas SyncPrefetch spends more than 300 seconds to accomplish the duplication process. When the table size increases to 10×10^5, SyncPrefetch's processing time is enlarged to 20 minutes, which is approximately 67 times longer than that of our GreenDB.

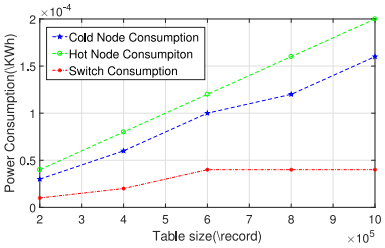
Energy consumption is a product of power and processing time, meaning that shortening the processing time is a practical way of boosting energy efficiency. Thus, reducing time spent in moving data tables from cold to hot nodes can minimize the energy consumption overhead of fetching hot data from cold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

5.3 Data Table Size

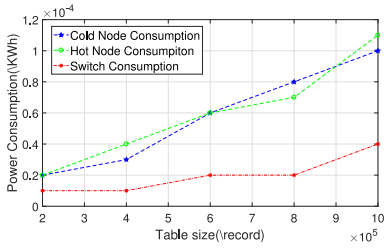
The second set of experiments is focused on evaluating the impacts of data table size (data size for short) on the energy savings offered by GreenDB after replicated tables are created on hot nodes. We set the number of cold nodes to four; we set the hit rate of the hot nodes to 25 percent (see Fig. 5a), 50 percent (see Fig. 5b), and 75 percent (see Fig. 5c), respectively. We compare our GreenDB with the two existing schemes - *PRE-BUD* [9] and *DPM* [34]. For comparison purpose, we also evaluate the database system where no energy-saving scheme is deployed; we refer to this case as *Non-Energy Saving* or *Non-E* for short.

Fig. 5 reveals that the data size imposes noticeable impact on the energy efficiency of the database cluster when the hit rate is lower than 75 percent. Unlike GreenDB and PRE-BUD, the energy consumption of DPM and Non-E are insensitive to data size. The data table size has very little impact on energy efficiency on DPM and Non-E, because power management in DPM and Non-E never creates replicated data tables from cold to hot nodes. Fig. 5 shows that increasing the data size make the energy consumption go up. This trend is contributed by the fact that if incoming queries are unable to be served by a hot node, corresponding cold nodes will be waked up to duplicate data. A large data size implies a long time spent in moving data tables to the hot node. Such a trend is diminishing when the hit rate is large (e.g., 75 percent), because the large hit rate cuts the chances of waking up cold nodes to move hot data.

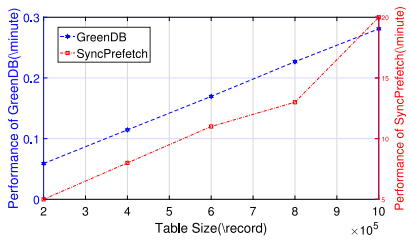
Fig. 5 shows that regardless of the data size, a large hot-node hit rate enables cold nodes to stay in the low-power mode for a long time period. This results suggest that a prefetching algorithm plays a key role in improving energy efficiency of cluster database systems. Such a prefetching mechanism will be addressed in our future study.



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

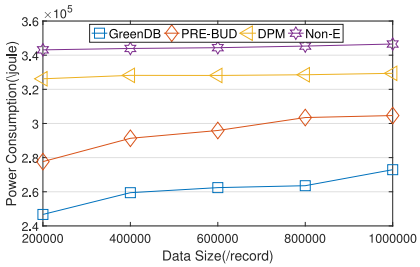


(b) Power consumption of replicate one data table from Cold node to Hot node

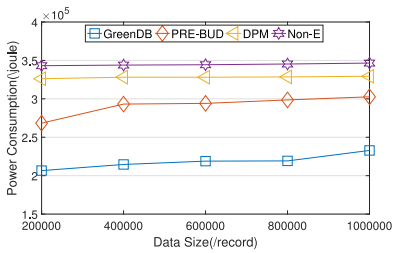


(c) Comparison of performance between GreenDB and SyncPrefetch

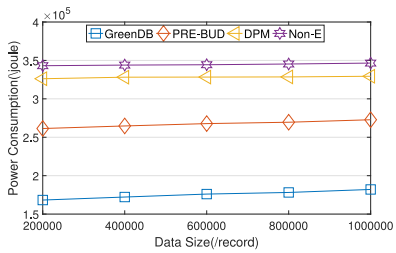
Fig. 4. Energy consumption and performance profiling between fetching and replicating techniques.



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



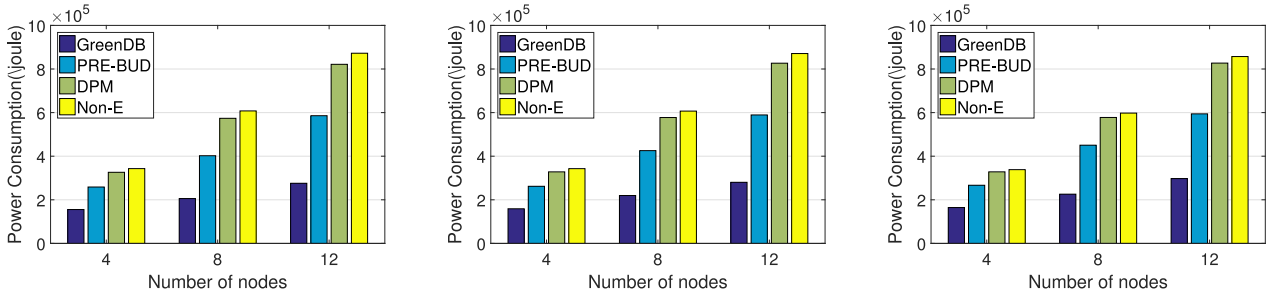
(b) Total energy consumption of DBMS when the hit rate is 50%.



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

Fig. 5. Total energy consumption of database clusters while data size is varied for three different values of the hit rate: (a) 25 percent, (b) 50 percent, (c) 75 percent.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

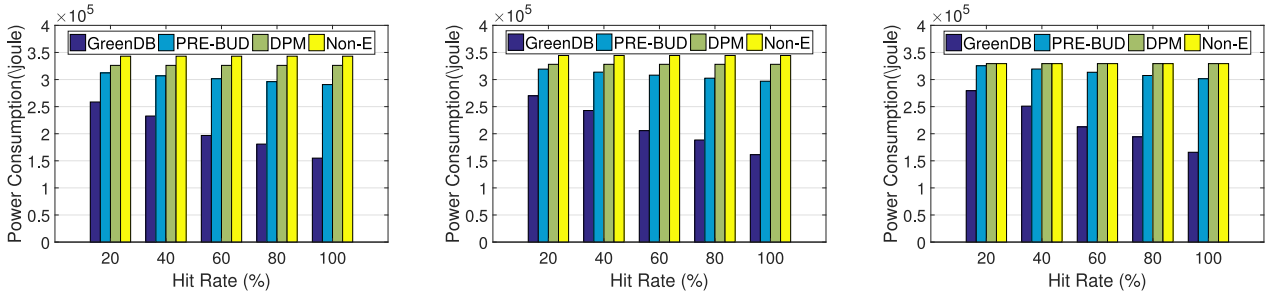


(a) Total energy consumption while data size is fixed at 200K records

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

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

Fig. 6. Total energy consumption of database clusters while the number of nodes is varied. Data size is fixed at (a) 200K, 400K, 800K records.



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

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

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

Fig. 7. Total energy consumption for different hit rate values where the data size is fixed at: (a) 200K records, (b) 400K records, (c) 600K records.

Fig. 5 indicates that the energy consumption of GreenDB increases slowly when hit rate reaches the level of 75 percent. GreenDB is superior to PRE-BUD, because PRE-BUD achieves high energy efficiency by aggressively turning off idle disks whereas GreenDB powered off the entire cold nodes rather than individual disks. In doing so, GreenDB conserves energy consumed by disks, CPUs, memory, and the other components in idle nodes. The experimental results confirm that small data table sizes coupled with high hit rates helps in improving energy efficiency of GreenDB.

5.4 The Number of Nodes

Now we evaluate the impact of the number of nodes on GreenDB. The number of hot node is fixed at one; the number of cold nodes is set to 4, 8, and 12, respectively. The hit rate is configured at the level of 85 percent. Fig. 6 shows the energy consumption of GreenDB, PRE-BUD, DPM, and Non-E under three data table sizes.

Not surprisingly, we discover from Fig. 6 that when the number of cold nodes goes up, energy savings offered by GreenDB becomes more pronounced. For example, let us consider a case where the data table size is 800K records (see Fig. 6c). The GreenDB reduces the energy consumption of Non-E by 51.4 and 65.3 percent when the number of the number of cold nodes is 4 and 12, respectively. This energy-efficiency trend is expected, because an increasing number of cold nodes are kept in the low-power mode to reduce energy consumption of the database system. After the hot node fetches popular tables from the cold nodes, lightly loaded cold data nodes are more likely to be switched into the power-saving mode to conserve energy by increase idle periods that are longer than the break-even time.

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

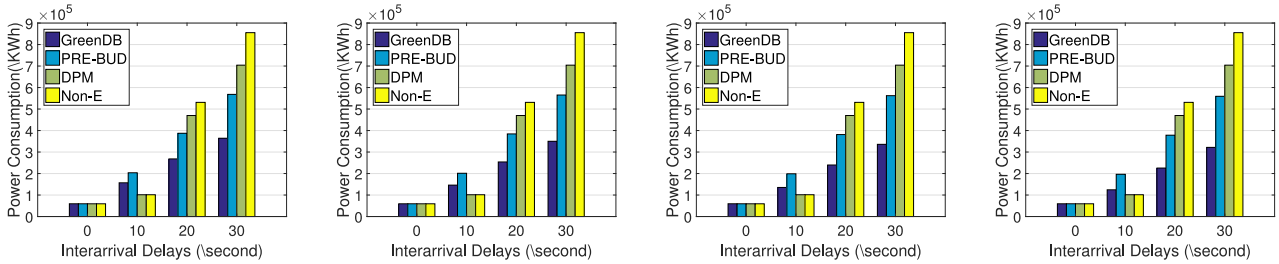
Comparing Fig. 6a and 6c, we observe that GreenDB deliver higher energy efficiency when data table size is larger (e.g., 800K records). This trend is consistent with the one depicted in Fig. 5 (see Section 5.3).

5.5 Hit Rates

Now we investigate the hit rate impact on the energy efficiency of the evaluated database system. Hit rate measures the probability that a query can be serviced by hot nodes rather than cold nodes. We configure the data table size at $200K$, $400K$ and $600K$ records. The number of cold nodes is set to four. Fig. 7 shows the total energy consumption as a function of hit rate values.

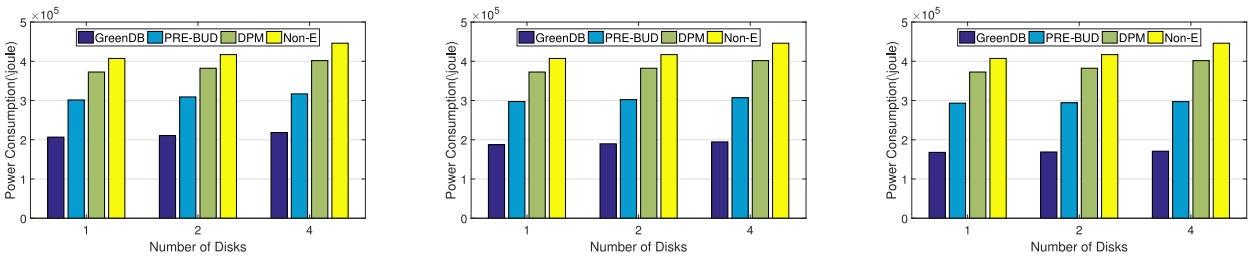
We observe from Fig. 7 that a high hit rate improves GreenDB's energy efficiency. For instance, when the hit rate is set to 80 and 20 percent, GreenDB reduces energy consumption of Non-E by 41.0 percent and 15.1 percent, respectively (see Fig. 7c). This trend is expected, because increasing hit rate enlarges the time period in which cold nodes are kept in the low-power state. A low hit rate leads to a strong likelihood that cold nodes are frequently waked up to serve requests. These experimental results are consistent with those plotted in Fig. 5. We argue that it is

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.



(a) Total energy consumption when the hit rate is : 55%. (b) Total energy consumption when the hit rate is : 65%. (c) Total energy consumption when the hit rate is : 75%. (d) Total energy consumption when the hit rate is : 85%.

Fig. 8. Total energy consumption for different delay values where the hit rate is : (a) 55 percent, (b) 65 percent, (c) 75 percent, and (d) 85 percent.



(a) Total energy consumption of database clusters when hit rate is fixed at: 60% (b) Total energy consumption of database clusters when hit rate is fixed at: 75% (c) Total energy consumption of database clusters when hit rate is fixed at: 90%

Fig. 9. Total energy consumption of database clusters while the number of data disks of cold node is varied. Hit rate is fixed at: (a) 60 percent, (b) 75 percent, (c) 90 percent.

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

5.6 Inter-Arrival Delays

Now we study the impact that query inter-arrival rates on the energy savings of GreenDB. Fig. 8 shows the total energy consumption of the database system as a function of the inter-arrival delay. Again, we set the number of cold nodes as four; the data size is fixed at 1,000,000 records. The hit rate is set to 55, 65, 75, and 85 percent, respectively.

Fig. 8 demonstrated that when the query inter-arrival delay is less than 10 seconds, GreenDB and PRE-BUD are unable to yield any energy savings. Such a low energy efficiency is attributed by the lack of large idle windows that keep cold nodes in the energy-saving state. Unfortunately, in these scenarios, GreenDB and PRE-BUD end up consuming more energy than the traditional DPM due to the prefetching overhead.

Both GreenDB and PRE-BUD start delivering energy savings when the inter-arrival delay increase. For example, Fig. 8d confirms that when the inter-arrival delay is as large as 30 seconds, GreenDB and PRE-BUD conserve energy by 57.4, 33.6 percent, respectively; DPM saves energy by a meager percentage of 17.7 percent. Intuitively, DPM relies on large idle times between consecutive queries to achieve energy savings; GreenDB, on the other hand, proactively creates large idle windows for cold nodes by redirecting queries to the hot node in the database system.

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

5.7 Number of Disks per Cold Node

Now we evaluate the capacity impact of cold nodes. In particular, we investigate whether or not increasing the number of disks in each cold node can optimize the energy efficiency of database systems running on clusters. Fig. 8 shows the energy consumption of the database system, where the number of disks in each node is set to one, two, and four. And the data size is fixed at $200K$ records; the hit rate is varied at 60, 75, and 90 percent, respectively.

Not surprisingly, we discover from Fig. 9 that increasing the number of disks in each node marginally improves the energy efficiency of GreenDB. For example, let us consider a case where hit rate is 75 percent (Fig. 9b). GreenDB reduces energy consumption of Non-E by 54.0 and 56.4 percent when the number of disks per node is one and four, respectively. If hit rate is 90 percent, then GreenDB's energy saving is 58.8 and 61.7 percent for 1-disk and 4-disk cases. The results indicate that adding two extra disks merely increases the energy savings by approximately 2.4 and 2.9 percent when hit rate is set to 75 and 90 percent, respectively.

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

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

solving this problem is to add disks into hot nodes when we expand cold nodes.

6 Conclusions

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

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

As a future research direction, we plan to extend GreenDB to the realm of NoSQL database clusters. We will investigate three types of NoSQL database systems, namely key-value database (e.g., Redis), column-oriented database (e.g., Cassandra), document database (e.g., MongoDB). We intend to focus on developing prefetching strategies for these three types of NoSQL databases running on clusters.

Acknowledgments

This work was supported in part by was supported by the U.S. National Science Foundation under Grants IIS-1618669, CNS-0917137, and CCF-0845257.

References

[1] P. Mahadevan, S. Banerjee, P. Sharma, A. Shah, and P. Ranganathan, "On energy efficiency for enterprise and data center networks," *IEEE Commun. Mag.*, vol. 49, no. 8, pp. 94–100, Aug. 2011.

[2] M. G. Xavier, I. C. De Oliveira, R. D. Dos Passos, and C. A. De Rose, "Towards better manageability of database clusters on cloud computing platforms," in *Proc. 29th Annu. ACM Symp. Appl. Comput.*, 2014, pp. 366–367.

[3] W. Rodiger, T. Muhlbauer, P. Unterbrunner, A. Reiser, A. Kemper, and T. Neumann, "Locality-sensitive operators for parallel main-memory database clusters," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 592–603.

[4] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso, "Scalable replication in database clusters," in *Distributed Computing*. New York, NY, USA: Springer, 2000, pp. 315–329.

[5] A. A. Lima, C. Furtado, P. Valduriez, and M. Mattoso, "Parallel olap query processing in database clusters with data replication," *Distrib. Parallel Databases*, vol. 25, no. 1–2, pp. 97–123, 2009.

[6] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman, "Parallel skyline queries," *Theory Comput. Syst.*, vol. 57, no. 4, pp. 1008–1037, 2015.

[7] A. Freire, "Method and system for query transformation for managing information from multiple datasets," U.S. Patent App. 14/171,712, Feb. 3, 2014.

[8] H. Saxena and K. Salem, "Edgex: Edge replication for web applications," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, 2015, pp. 1041–1044.

[9] A. Manzanares, X. Qin, X. Ruan, and S. Yin, "Pre-bud: Prefetching for energy-efficient parallel i/o systems with buffer disks," *ACM Trans. Storage*, vol. 7, no. 1, 2011, Art. no. 3.

[10] C. Patel and P. Ranganathan, "Enterprise power and cooling," *ASPLOS Tutorial*, Oct. 2006.

[11] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, "Ead and pebd: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 360–374, Mar. 2011.

[12] A. Beloglazov and R. Buyya, "Openstack neat: A framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds," *Concurrency Comput.: Practice Exp.*, vol. 27, no. 5, pp. 1310–1333, 2015.

[13] J. Kołodziej, T. Burczyński, and A. Y. Zomaya, "A note on energy efficient data, services and memory management in big data information systems," *Inf. Sci.: An Int. J.*, vol. 319, no. C, pp. 69–70, 2015.

[14] Z. Yang, L. Liu, C. Qiao, S. Das, R. Ramesh, and A. Y. Du, "Availability-aware energy-efficient virtual machine placement," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 5853–5858.

[15] Y. Chehadeh, A. R. Hurson, and L. L. Miller, "Energy-efficient indexing on a broadcast channel in a mobile database access system," in *Proc. Int. Conf. Inf. Technol.: Coding Comput.*, 2000, pp. 368–374.

[16] D. C. Cromer, B. J. Ellison, E. R. Kern, and H. Locker, "Data processing system and method for waking a client only in response to receipt of an authenticated wake-on-lan packet," U.S. Patent 6,526,507, Feb. 25, 2003.

[17] M. Ghamkhari and H. Mohsenian-Rad, "Energy and performance management of green data centers: A profit maximization approach," *IEEE Trans. Smart Grid*, vol. 4, no. 2, pp. 1017–1025, Jun. 2013.

[18] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.

[19] L. Wang, F. Zhang, J. A. Aroca, A. V. Vasilakos, K. Zheng, C. Hou, D. Li, and Z. Liu, "Greendcn: A general framework for achieving energy efficiency in data center networks," *IEEE J. Select. Areas Commun.*, vol. 32, no. 1, pp. 4–15, Jan. 2014.

[20] N. Nishikawa, M. Nakano, and M. Kitsuregawa, "Energy efficient storage management cooperated with large data intensive applications," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 126–137.

[21] 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 *Proc. 16th Int. Conf. Comput. Commun. Netw.*, 2007, pp. 735–740.

[22] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[23] G. Graefe, "New algorithms for join and grouping operations," *Comput. Sci.-Res. Develop.*, vol. 27, no. 1, pp. 3–27, 2012.

[24] D. Schall and T. Härder, "Towards an energy-proportional storage system using a cluster of wimpy nodes," in *Proc. BTW*, 2013, pp. 311–325.

[25] Y. Zhou, S. Taneja, X. Qin, W.-S. Ku, and J. Zhang, "EDOM: Improving energy efficiency of database operations on multicore servers," *Future Generation Comput. Syst.*, Mar. 2017.

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

[27] J. Shuja, K. Bilal, S. A. Madani, M. Othman, R. Ranjan, P. Balaji, and S. U. Khan, "Survey of techniques and architectures for designing energy-efficient data centers," *IEEE Syst. J.*, vol. 10, no. 2, pp. 507–519, Jun. 2016.

[28] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, R. Y. Wang, et al., "Modeling hard-disk power consumption," in *Proc. FAST*, 2003, vol. 3, pp. 217–230.

[29] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues, "A uto p lacer: Scalable self-tuning data placement in distributed key-value stores," *ACM Trans. Auton. Adaptive Syst.*, vol. 9, no. 4, 2015, Art. no. 19.

[30] N. Diegues and P. Romano, "Self-tuning intel transactional synchronization extensions," in *Proc. Int. Conf. Autonomic Comput.*, 2014, pp. 209–219.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.

[31] Z. Xu, C. Stewart, N. Deng, and X. Wang, "Blending on-demand and spot instances to lower costs for in-memory storage," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[32] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 231–242.
[33] R. Ramakrishnan, and G. Johannes, *Database Management Systems*. McGraw Hill, 2000.
[34] J. Kim, M. Ruggiero, and D. Atienza, "Free cooling-aware dynamic power management for green datacenters," in *Proc. Int. Conf. High Perform. Comput. Simul.*, 2012, pp. 140–146.



Yi Zhou received the BEE and MSEE degrees in electronic engineering from Beijing University of Technology, Beijing, in 2006 and 2010, respectively, and the PhD degree in computer science from Auburn University, in 2018. He is currently an assistant professor with TSYS School of Computer Science, Columbus State University. Prior to joining Columbus State University in 2018, he has been a software engineer with Alcatel-Lucent Technologies (China) Co.,Ltd. for four years from 2010 to 2014. His research interests include energy-saving techniques, database systems, and parallel computing.



Shubbhi Taneja received the BE 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. She is an Assistant Professor of Computer Science at Sonoma State University. Her primary research area is energy-efficient computing, with a focus on large-scale datacenters and supercomputers. She is also interested in high-performance computing and computer science education. Her main area of research include parallel and distributed systems, storage systems, energy-efficient computing, and performance evaluation. She is a member of IEEE, ACM, and IEEE Computer Society.



Chaowei Zhang received the BE degree from the North University of China, in 2013 and joined Samuel Ginn College of Engineering, Auburn University, to pursue the doctoral degree of computer science in 2016. His research area includes data mining, parallel and distributed systems, deep learning and natural language processing..



Xiao Qin (S'00-'04M-SM'09) received the BS and MS degrees in computer science from Huazhong University of Science and Technology, in 1992 and 1999, respectively, and the PhD degree in computer science from the University of Nebraska-Lincoln, in 2004. He is currently a professor with the Department of Computer Science and Software Engineering, Auburn University. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He received a 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. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:36:15 UTC from IEEE Xplore. Restrictions apply.