Noname manuscript No.
(will be inserted by the editor)

Enhancing Timeliness and Saving Power in Real-Time Databases

Kyoung-Don Kang

Received: date / Accepted: date

Abstract In data-intensive real-time embedded applications, it is desirable to process data service requests in a timely manner using fresh data, spending less power. However, related work is relatively scarce despite the importance. In this paper, we present an effective approach to reduce both deadline misses and power expenditure in real-time databases with one or more processor by merging similar real-time queries to decrease repeated data accesses and processing, while doing dynamic power management. In a simulation study, our approach substantially decreases deadline misses and power consumption compared to state-of-the-art baselines.

Keywords Real-Time Databases · Timeliness · Power Conservation · Query Aggregation

1 Introduction

The demand for real-time data services in embedded systems is increasing. For example, small-footprint embedded databases are developed to support data-intensive real-time embedded and cyber-physical applications, such as traffic control, surveillance, smart buildings, energy-efficient avionics, medical devices, firefighting, and real-time engine diagnosis [mcobject(2017), Kang and Chung(2017), Kang and Son(2012), Kang and Chung(2015), Gustafsson et al(2005)Gustafsson, Hallqvist, and Hansson].

In data-intensive real-time embedded applications, it is desirable yet challenging to process real-time transactions and queries (read-only transactions)

Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
Tel.: 1-607-777-4368
E-mail: kang@binghamton.edu

in a timely manner using fresh data representing the current real-world status, while reducing the power consumption. Optimal energy scheduling is intractable even in a single processor with dynamic voltage and frequency scaling (DVFS) and a low-power idle state [Irani et al(2007)Irani, Shukla, and Gupta]. Saving power without impairing the timeliness in real-time databases (RTDBs) could be more challenging, because the arrival rate of user transactions may vary considerably depending on the current real-world status, e.g., the traffic or weather conditions [Lam and Kuo(2006)]. Further, transactions may get aborted and restarted due to data/resource conflicts unknown *a priori*, incurring deadline misses. Therefore, in this paper, we investigate effective heuristics to decrease both the deadline miss ratio and power consumption in RTDBs with one or more processor, while supporting the required data freshness.

In a database system, multiple queries often access common data, e.g., traffic data of busy intersections or severe weather data. In fact, large data access skews are prevalent [Wires et al(2014)Wires, Ingram, Drudi, Harvey, and Warfield]. For example, the well-known 80/20 rule indicates that 20% of data are accessed for 80% of accesses. However, a direct application of this approach to RTDBs may result in many deadline misses, because queries should be delayed for aggregation. For instance, Lang et al. [Lang and Patel(2009)] intentionally delay queries to combine them in non-RTDBs. Although their approach decreased the energy consumption by up to 54%, it increased the average response time by 43%.

To combine real-time queries that access common data without jeopardizing the timeliness, we schedule the transaction with the highest priority (e.g., the earliest deadline transaction) first, while scanning the ready queue sorted in non-ascending priority order backwards to merge similar user transactions together in the background.[1] In this way, we avoid duplicate data analyses as well as accesses in RTDBs as much as possible without disrupting high priority transactions at or near the head of the queue. In traffic control, for example, several queries for travel route planning may require the RTDB to read and process the same sensor data by running a shortest path algorithm to find a path from location A to B expected to be fastest considering the geographic distance and current traffic status, even if their sources and final destinations are different. Multiple real-time queries may require the RTDB to access and process common data to analyze the air quality and micro-climate conditions in a smart building to process queries for allergy warnings or heating, ventilation, and air conditioning (HVAC). Also, several real-time surveillance queries may need to access and process common surveillance images captured on a busy street to do background subtraction and object detection, even if they are tracking different objects, e.g., different types of cars or pedestrians, in the images. Thus, our approach for real-time query aggregation combines multiple queries into a single query to avoid repeated data accesses and analyses of common data, which may incur significant overhead and deadline misses,

[1] Although we not only aggregate queries but also combine data reads and processing for transactions, we call our approach real-time query aggregation to be consistent with the term 'query aggregation' used in the database literature [Lang and Patel(2009)].

while reducing data/resource contention among transactions that could cause transaction aborts/restarts.

Race-to-idle and never-idle are two major approaches for power saving. In the race-to-idle scheme, the processor runs at the maximum speed to enter a low-power idle state as early as possible. In the never-idle scheme, however, the processor speed is continuously adapted to meet the performance requirement with less power consumption. In this paper, we mainly take a race-to-idle approach to reduce the processor power consumption due to the decreasing effectiveness of the never-idle approach based on, for example, DVFS [Bambagini et al(2016)Bambagini, Marinoni, Aydin, and Buttazzo]. At runtime, our approach processes real-time update and user transactions at the highest processor speed, while reducing the RTDB workload via real-time query aggregation. When a processor is idle with no update or user transaction to execute, our approach switches it to a low-power idle mode based on the idle interval length estimated considering the data update periods and recent user transaction arrival pattern. Thus, our real-time query aggregation and power saving methods cooperate with each other to simultaneously reduce both the deadline miss ratio and power consumption rather than doing trade-offs between them.

Despite the importance, related work on RTDB power management is relatively scarce [Kang and Chung(2015), Kang and Son(2012)]. A summary of our key contributions and novelty follows:

- Our real-time query aggregation scheme reduces both the miss ratio and power consumption rather than doing trade-offs between them.
- It is generally applicable to RTDB power management, since it does not assume a constrained or specialized transaction/query model.
- It requires neither any special hardware nor extensive system modeling and tuning. It only needs low-power idle states supported by almost all processors today.
- Our approach is configurable and relatively easy to use. A database administrator (DBA) needs to set only a few parameters for real-time query aggregation.

In this paper, we have considerably extended our previous work [Kang(2016)], which only considers uniprocessor RTDBs, to reduce the deadline miss ratio and power consumption in multiprocessor as well as uniprocessor RTDBs:

- We extend the RTDB system architecture to provide fundamental support for power-aware real-time data services in RTDBs with one or more processors, e.g., real-time transaction scheduling and power management.
- Based on the new RTDB architecture with $m \geq 1$ processors, we extend the real-time query aggregation and dynamic power management (DPM) schemes. Especially, we aggregate real-time queries by scanning the global earliest deadline first (GEDF) queue backwards. We assign a set of aggregated real-time queries together to one processor in the EDF order to share the common data and processing results already retrieved and produced by earlier deadline transactions, if any. In this way, we reduce repeated

memory accesses and computations, which may occur otherwise, to decrease the deadline miss ratio, while saving power via DPM. Our approach naturally maps to a uniprocessor RTDB too: In a uniprocessor RTDB, we schedule transactions via EDF and scan the EDF queue backwards to decrease duplicate data accesses and analyses to reduce the miss ratio and power consumption via real-time query aggregation and DPM. Thus, our approach conserves power for relatively moderate workloads, while enhancing the timeliness for higher workloads in RTDBs with one or more processor.

– We support the data freshness requirements based on the notion of absolute validity intervals (discussed in Section 3) [Lam and Kuo(2006)] without dynamically adapting the freshness different from our previous work [Kang(2016)]. Although dynamic freshness adaptation could be effective for certain applications, e.g., weather or stock price updates on the web with relatively relaxed freshness requirements, freshness assurance based on absolute validity intervals can be more appropriate for data-intensive real-time embedded or cyber-physical applications such as traffic control or surveillance, in which stringent freshness requirements are determined by physical characteristics.
– In addition, we have thoroughly evaluated the performance of our approach in not only uniprocessor but also multiprocessor RTDBs via an extensive simulation study modeled after real-world RTDB applications, e.g., traffic control, fire detection, and engine diagnosis [Xiong et al(2008)Xiong, Han, Lam, and Chen, Han et al(2014)Han, Chen, Xiong, Lam, Mok, and Ramamritham, Kang and Son(2012)]. Our approach decreases the deadline miss ratio and dynamic power consumption compared to the tested state-of-the-art baselines by up to approximately 86% and 45% for different workload and numbers of cores (1−8 in Section 5).

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 discusses the supported transaction types, data freshness requirements, and power-aware RTDB architecture. Section 4 describes our approach for reducing the deadline miss ratio and power consumption in RTDBs. Section 5 evaluates the performance of our approach and baselines. Section 6 discusses open research issues. Finally, Section 7 concludes the paper.

2 Related Work

In general, research on power/energy management in database systems is relatively new. It is known that [Lang and Patel(2009)] is the first to provide concrete techniques for energy-efficient query processing. It explicitly delays queries for combined processing, while supporting DVFS. It has been followed by other projects on database power/energy management in data centers including [Xu et al(2013)Xu, Wang, and Tu, Tu et al(2014)Tu, Wang, Zeng, and Xu, Kunjir et al(2012)Kunjir, Birwa, and Haritsa, Xu et al(2015)Xu, Tu, and Wang]. In these approaches, the database energy consumption is reduced for the increased response time or decreased throughput. However, they are not directly applicable to RTDBs, since delaying real-time queries to enhance energy efficiency via aggregation may incur many deadline misses as discussed before. Neither do they support real-time transaction scheduling, concurrency control, or data freshness (temporal consistency). Sensor network databases [Madden et al(2005)Madden, Franklin, Hellerstein, and Hong,Tsiftes and Dunkels(2011)] support relatively simple in-network data processing, e.g., sensor data aggregation, to mainly optimize communication costs. In [Hu and et al.(2015)], efficient data freshness management is explored when data are retrieved from wireless sensors in sequence specialized for rescue or tactical situations. Kim et al. [Kim et al(2016)Kim, Abdelzaher, Sha, Bar-Noy, and Hobbs] propose a scheduling algorithm to retrieve sensor data over a single bottleneck connection, which is optimal on the condition that no data is shared among real-time decision making tasks. However, aggregating real-time queries and reducing the RTDB power consumption are not considered.

Surprisingly little work has been done on RTDB power management. A novel work [Kang and Son(2012)] is the first to support the desired power consumption and I/O deadline miss ratio, via multi-input, multi-output (MIMO) control, in an RTDB based on flash memory. In [Kang and Chung(2015)], a control theoretic approach is developed to support the timeliness of a single periodic real-time transaction running concurrently with a few interfering non-real-time transactions in an embedded database. The power consumption is decreased via dynamic frequency scaling and sensor data dropping in the feedback loop. In [Kang and Chung(2017)], the same authors support DVFS, while adapting the data temporal consistency for fine-grained control in the feedback loop despite discrete voltage/frequency levels. However, the general applicability of [Kang and Chung(2015),Kang and Chung(2017)] is limited, because they have a constrained transaction model that supports one real-time user transaction only. In [Xu et al(2013)Xu, Wang, and Tu], DVFS based on proportional and integral (PI) control is applied to decrease the power consumption for I/O bound queries, while supporting the desired throughput in a non-RTDB. The energy costs of query operators are evaluated in [Xu et al(2015)Xu, Tu, and Wang]. However, real-time query aggregation is not considered to reduce the RTDB power consumption. These projects [Kang and Son(2012), Kang and Chung(2015), Kang and Chung(2017), Xu et al(2013)Xu, Wang, and Tu] essentially take never-idle approaches that require extensive system modeling and tuning, which should be repeated in different platforms. Our approach adopts a more general real-time transaction and data model. Also, it reduces RTDB power consumption via DPM without requiring complex modeling and tuning.

Power-aware real-time scheduling has been well explored. Related work includes [Irani et al(2007)Irani, Shukla, and Gupta, Völp et al(2014)Völp, Hähnel, and Lackorzynski, Cao and Ravindran(2014), Imes et al(2015)Imes, Kim, Maggio, and Hoffmann, Legout et al(2015)Legout, Jan, and Pautet, Fu et al(2016)Fu, Calinescuy, Wang, Li, and Xue, D'souza and Rajkumar(2017), Guo et al(2017)Guo, Bhuiyan, Saifullah, Guan, and Xiong,Kehr et al(2017)Kehr, Quinones, Langen, Boeddeker, and Schaefer] just to name a few. A good survey of power management in hard real-time systems is given in [Bambagini et al(2016)Bambagini, Marinoni, Aydin, and Buttazzo]. Although the transaction timeliness, data freshness, and power management in RTDBs are not directly considered in these approaches, basic principles could be applied for more effective RTDB power management.

Novel hardware and operating system techniques are developed to deal with interference among the CPU cores in safety-critical real-time embedded systems [Valsan et al(2017)Valsan, Yun, and Farshchi, Kim et al(2017)Kim, Ward, Chisholm, Fu, Anderson, and Smith]. Our approach is complementary to them in that it strives to reduce user transaction workloads and power consumption via real-time query aggregation and DPM, potentially decreasing interference by executing aggregated real-time queries in the same core.

Further, our work could benefit more from previous database research. For example, approximate query processing techniques are developed to produce rough results under overload [Deshpande et al(2007)Deshpande, Ives, and Raman, Babu and Bizarro(2005)]. The miss ratio and power consumption of our approach could be decreased further, if it is combined with approximate query processing. In [Xiong et al(2008)Xiong, Han, Lam, and Chen, Han et al(2014)Han, Chen, Xiong, Lam, Mok, and Ramamritham], update transactions are deferred to reduce the update workload, supporting the data temporal consistency. Little prior work has been done to support the freshness of temporal data in multiprocessor environments. In [Li et al(2011)Li, Chen, Xiong, and Li], a novel method is developed to maintain the data temporal consistency via workload-aware partitioning under EDF. Han et al. [Han et al(2016)Han, Lam, Chen, Xiong, Wang, Ramamritham, and Mok] present novel algorithms to maintain data temporal consistency even during a mode change in dynamic cyber-physical systems with multi-modal behavior. Thus, our work is complementary to these approaches.

3 Data Types, Transactions, and System Overview

In this section, the data and transaction types and data freshness requirements considered in this paper are described. Background for DPM is given. Also, an overview of our power-aware RTDB architecture is given.

3.1 Data Types, Transactions, and Deadlines

In our data service model, there are two types of data: *temporal* and *non-temporal* data. Temporal data, e.g., sensor readings, become outdated as time goes by, because the real-world status, e.g., traffic or weather state, may continuously change. The temporal consistency between the real-world state and the temporal data in the RTDB is maintained based on the *absolute validity intervals* [Lam and Kuo(2006)]. A temporal data item O_i is associated

```
<begin transaction $T_i$>
   read and analyze $O_i$;
   read and analyze $O_j$;
     ...
   write $O_k$;
   deadline $D_i$; /* relative deadline */
<end transaction $T_i$>
```

Fig. 1 An Example of a Real-Time Transaction T_i

with a timestamp that indicates the latest update time. It is considered fresh, i.e., temporally consistent, if (current time − timestamp$(O_i) \leq avi(O_i)$) where $avi(O_i)$ is the absolute validity interval of O_i. On the other hand, non-temporal data, e.g., a vehicle registration number, do not become outdated unless users explicitly modify them. Thus, we focus on managing temporal data in this paper.

In RTDBs, there are two types of transactions. If a real-time transaction T_i is a *periodic update transaction*, T_i periodically updates a specific sensor data object O_i in a dedicated manner. Also, a user can submit an *aperiodic user transaction* to the RTDB to read temporal data and read/write non-temporal data to derive real-time information from raw sensor data [Lam and Kuo(2006)]. If T_i is a user transaction, it can read and process sensor data periodically refreshed by the update transactions to support, for example, transportation management or surveillance.

In general, a real-time transaction T_i reads and processes a set of data R_i and writes a set of data W_i as illustrated in Figure 1. If T_i is an update transaction for O_i, the read set $R_i = \emptyset$ and the write set $W_i = \{O_i\}$. On the other hand, if T_i is a user transaction, R_i consists of one or more temporal/non-temporal data. W_i consists of zero or more non-temporal data. (If T_i is a query, $W_i = \emptyset$.)

If T_i is an update transaction, its relative deadline is equal to its period, i.e., $D_i = P_i$. In contrast, the relative deadlines of user transactions are determined by a specific RTDB application of interest, e.g., transportation management or surveillance. If an update or user transaction T_i with a relative deadline D_i is released or arrives at time t, its absolute deadline is $t + D_i$. In this paper, real-time transactions are assigned *firm* deadlines. If all the required operations to read, process, and write data are completed by $t + D_i$, T_i is committed successfully. Otherwise, it is aborted upon the deadline miss to avoid cascading deadline misses due to intensified data/resource contention.

3.2 Background for Dynamic Power Management

For RTDB power management, we consider the advanced configuration and power interface (ACPI) standard that is widely adopted. In ACPI, P states are performance states. The P0 state supports the highest frequency and voltage. A higher numbered P state spends less power, but provides a lower computational speed due to the reduced frequency and voltage.

In addition, ACPI has C states. In the C0 state (active mode), the processor executes instructions normally. However, no instruction is executed in a low-power idle state, e.g., the C1, C2, or C3 state in Table 1, or during a state transition.[2] In this paper, we assume that the operating system provides an application programming interface (API) to let the RTDB leverage the low-power states to support timely, power-efficient real-time data services using ACPI when one or more core becomes idle.

Table 1 C-states used in this paper (source: [Legout et al(2015)Legout, Jan, and Pautet])

State (C_j)	**Power** (ρ_j)	**Latency** (δ_j)	**Energy** (E_j)
C0 (Run)	1 W	0	0
C1 (Standby)	0.5 W	0.1 ms	0.025 mJ
C2 (Dormant)	0.1 W	2 ms	0.9 mJ
C3 (Shutdown)	0.00001 W	10 ms	5 mJ

As shown in Table 1, a transition between the C0 state and C1 state takes relatively negligible time and energy. More power is saved in a higher C state; however, the state transition takes more time and energy. In the table, the power consumption ρ_j in the state C_j where $j > 0$ is normalized to that in the C0 state.

To effectively exploit the C-states, we define the transition latency of C_j as: $\delta_j = \delta_{0\to j} + \delta_{j\to 0}$ where $\delta_{0\to j}$ is the state transition latency from C0 to C_j and $\delta_{j\to 0}$ is that from C_j to C0. Also, we define the energy overhead for C_j as: $E_j = E_{0\to j} + E_{j\to 0}$ where $E_{0\to j}$ indicates the energy consumed to switch from C0 to C_j and $E_{j\to 0}$ is the energy spent to shift back from C_j to C0. In this paper, we define the *break-even time* $B_j = \delta_j$ for a low-power state j; that is, an idle interval must be at least as long as δ_j to effectively leverage C_j [Legout et al(2015)Legout, Jan, and Pautet,Bambagini et al(2016)Bambagini, Marinoni, Aydin, and Buttazzo].

In this paper, we assume that the same amount of power is consumed, if the same RTDB workload is executed in two different cores and no power saving technique is applied in any of the cores. In addition, we make the following assumptions:

– Different cores can be in different C-states or P-states at the same time.

[2] We have adopted Table 1 from a novel work on energy-efficient real-time scheduling [Legout et al(2015)Legout, Jan, and Pautet], which has derived the low-power state model summarized in the table by analyzing the ARM Cortex-A family processors and FreeScale Power architecture.

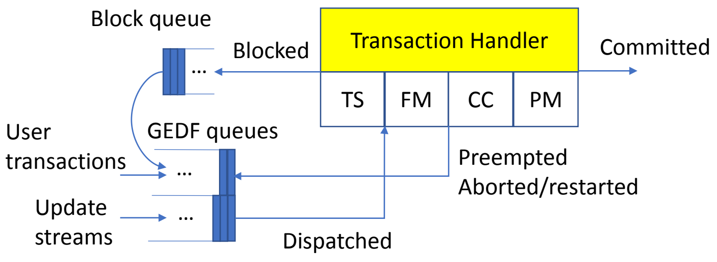


Fig. 2 Power-Aware Multicore RTDB Architecture

- There are a fixed number of discrete clock frequencies that range between $[f_{min}, f_{max}]$.[3]
- An idle core consumes more power in the highest P-state using the lowest frequency and voltage than it does in the C1 state that is the shallowest low-power mode.

Further, to model power saving due to real-time query aggregation and DPM, let us suppose that a core becomes idle after processing aggregated real-time queries assigned to it by TS in Figure 2. Also, suppose that PM in Figure 2 allows the idle core to switch to and stay in the C_j $(j > 0)$ state for T time units. In such a case, we estimate the average normalized power saving with respect to ρ_0 due to real-time query aggregation and DPM as follows:

1. We compute $p_s[t] = (\rho_0 - \rho_j)$W at each time instant t while the core is in the C_j state.
2. We then calculate the average normalized power saving considering E_j that is the energy overhead of the C_j state in Table 1:

$$p'_s[t] = \frac{1}{\rho_0 T}\left[\sum_{t=1}^{T} p_s[t] - E_j\right] (\%) \tag{1}$$

3.3 Power-Aware RTDB Architecture

The transaction handler in Figure 2 consists of the transaction scheduler (TS), freshness manager (FM), concurrency controller (CC), and power manager (PM). TS schedules real-time user transactions and data updates. In this paper, we apply the global earliest deadline first (GEDF) algorithm, because GEDF supports bounded tardiness in a multiprocessor real-time system with m processors, if the total utilization does not exceed m and the utilization of each processor is not higher than 1 [Devi and Anderson(2008)]. In our RTDB architecture, when $m = 1$, GEDF reduces to EDF as discussed before. Our

[3] The dynamic power consumption when the CPU is in the active mode is proportional to the clock frequency f and the square of the supply voltage V: $P_{dyn} \propto fV^2$ [Bambagini et al(2016)Bambagini, Marinoni, Aydin, and Buttazzo].

approach aggregates real-time user queries by scanning the EDF queue backwards and applies DPM to decrease the miss ratio and power consumption.

It is known that the runtime overhead due to task migration in GEDF is negligible for relatively small m [Bastoni et al(2010)Bastoni, Brandenburg, and Anderson]. Further, most existing embedded RTDBs run on uniprocessor or multicore platforms with a few cores [mcobject(2017),Kang and Chung(2017), Kang and Son(2012),Kang and Chung(2015),Gustafsson et al(2005)Gustafsson, Hallqvist, and Hansson, Lam and Kuo(2006)]. Similarly, we assume that the RTDB depicted in Figure 2 runs on a platform with $m \geq 1$ homogeneous core(s) that can be in different C-states. Further, we assume that a user or update transaction is processed sequentially in one of the cores (no intra-transaction parallelism).

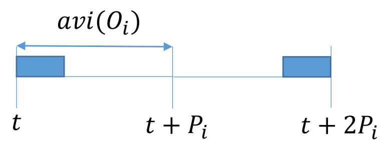


Fig. 3 One-One Principle: O_i becomes stale after $t + P_i$.

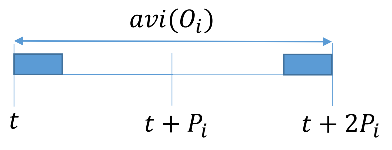


Fig. 4 Half-Half Principle: O_i is maintained fresh.

Maintaining the freshness of temporal data is important, because real-time user transactions may make wrong decisions, e.g., traffic control, HVAC, or emergency building evacuation decisions, based on stale data. A common misconception is that the freshness of a temporal data object O_i is supported by setting $P_i = avi(O_i)$ and meeting all update transaction deadlines [Stankovic et al(1999)Stankovic, Son, and Hansson]. In fact, this principle, called the one-one principle, may fail to ensure the data freshness even when all update transactions commit within their deadlines. For example, in Figure 3, two periodic jobs for updating O_i are scheduled. Although the second job is preempted by some higher priority transactions, both of them meet their deadlines. In this example, however, O_i becomes stale after $t+P_i$. In contrast, the half-half principle, which requires $P_i = 0.5 \times avi(O_i)$, guarantees the data freshness as long as the update transaction deadlines are met [Stankovic et al(1999)Stankovic, Son, and Hansson] as illustrated in Figure 4. Thus, we apply the half-half principle to meet the data freshness requirements in this paper.

In Figure 2, TS uses two separate GEDF queues to schedule user and update transactions, respectively. Higher priority is given to update transactions, similar to [Lam and Kuo(2006),Xiong et al(2008)Xiong, Han, Lam, and Chen, Han et al(2014)Han, Chen, Xiong, Lam, Mok, and Ramamritham, Stankovic et al(1999)Stankovic, Son, and Hansson, Han et al(2014)Han, Chen, Xiong, Lam, Mok, and Ramamritham, Kang and Son(2012), Kang and Chung(2015), Kang and Chung(2017)]. In this paper, we take this approach and make the following assumptions to maintain the data freshness in multiprocessor RTDBs, because processing real-time user transactions using stale data could be meaningless or even harmful in data-intensive real-time embedded applications, e.g., traffic control or surveillance:

- When there are N temporal data items in the RTDB, we assume that the period, P_i, and worst-case execution time, E_i, of the update task that periodically updates the temporal data object O_i $(1 \leq i \leq N)$ are known a priori and the relative deadline $D_i = P_i$ (implicit deadline).
- We assume that the set of all the update transactions in the RTDB meets the following properties:
  - Property 1. $U_{sum} = \sum_{i=1}^{N} \frac{E_i}{P_i} \leq (m+1)/2$
  - Property 2. $U_{max} = max_{i=1}^{N}(\frac{E_i}{P_i}) \leq 1/2$

  Under this assumption, the RTDB can meet all update transaction deadlines and the data freshness requirements enforced by FM in Figure 2, since GEDF can meet all deadlines of any implicit-deadline sporadic task system that satisfies the two properties on m processors (page 78 in [Baruah et al(2014)Baruah, Bertogna, and Buttazzo]). (In Section 5, U_{sum} and U_{max} of our update transactions for performance evaluation are approximately $m/2$ and 6%, respectively.)

However, supporting the timeliness of user transactions is challenging, because they are given lower priority and their arrival rate may vary significantly due to, for example, unpredictable traffic incidents or HVAC component failures. Also, they can get aborted/restarted due to unpredictable data/resource contention [Stankovic et al(1999)Stankovic, Son, and Hansson, Ramamritham et al(2004)Ramamritham, Son, and DiPippo, Lam and Kuo(2006), Kang and Son(2012), Kang and Chung(2015), Kang and Chung(2017), Kang(2016)]. Thus, in this paper, we mainly focus on reducing the deadline miss ratio of user transactions, while decreasing the power consumption in RTDBs via real-time query aggregation and DPM performed by TS and PM, respectively.

CC supports the serializability of concurrent transactions. For concurrency control, we support the two phase locking with high priority (2PL-HP) scheme [Lam and Kuo(2006)]. A data conflict arises in a database system, if two transactions access the same data item and at least one of them needs to write it. Under 2PL-HP, a low priority transaction is aborted and restarted upon a data conflict. However, it gets blocked, if it is requesting a data item already locked by a higher priority transaction in an incompatible manner. A restarted or blocked transaction is moved to the block queue. It is inserted back into the GEDF ready queue when the conflicting higher priority transaction(s) commit(s) as illustrated in Figure 2.

4 Decreasing Deadline Misses and Power Consumption in RTDBs

Our approach begins to run when the RTDB is initialized. It continues to run until either the DBA explicitly turns it off or shuts the system down. In this section, we give an overview and detailed descriptions of our approaches for real-time query aggregation and DPM to reduce the miss ratio and power consumption in RTDBs.

4.1 Overall Approach

Algorithm 1: Power-Aware Real-Time Data Services

```
while true do
    if a user transaction $T_i$ arrives at time $t$ then
        insert($T_i$, GEDF_QUEUE);
        call RTQA($T_i$) /*Algorithm 2: Real-Time Query Aggregation*/ ;
    Schedule real-time transactions;
    for $j = 1; j \leq m; j{+}{+}$ do
        if core $j$ is idle then
            switch core $j$ to a lower-power mode /*Algorithm 3: DPM */;
```

Algorithm 1 gives an overview of our approach. When a user transaction arrives, it is inserted into a specific position in the GEDF queue for user transactions in Figure 2 based on its deadline. In this paper, the user transaction at the i^{th} position in the GEDF queue is referred to as T_i. Thus, T_0 is the user transaction with the earliest deadline.

In Lines 2−4 of Algorithm 1, we support *lightweight, incremental* aggregation of real-time user transactions. Our real-time query aggregation method, Algorithm 2 (Section 4.2), attempts to aggregate T_i with the transaction(s) in front of it when it is inserted into the i^{th} (≥ 1) place in the GEDF queue as illustrated in Figure 5. Thus, a user transaction with a longer deadline is likely to be aggregated with more user transactions with shorter deadlines. By doing this, we intend to reduce the user transaction load without disrupting transactions with imminent deadlines.

In Line 5, TS in Figure 2 *assigns a series of real-time user transactions aggregated in the GEDF queue to one of the available cores in an EDF order.* TS first assigns the user transaction with the earliest deadline in the group, e.g., the first red transaction in Figure 5, to an available core (if any). The first transaction in the group accesses and processes the common data plus any other data that it needs to commit. The following user transactions in the group, e.g., the second and third red ones in Figure 5, run in the same core in an EDF order to share the common data and processing results as much as possible. If some of the queries with later deadlines in the series are preempted or aborted/restarted by a higher priority transaction and reinserted into the GEDF queue, Algorithm 2 merges them with the adjacent real-time queries (if any) in the queue depending on their data access needs to reduce repeated data accesses and processing.

In Lines 6−8, PM switches each core that is idle after TS schedules all user and update transactions to an appropriate low-power mode selected by Algorithm 3 (Section 4.3). In our RTDB architecture in Figure 2, TS needs to wake up an idle core in a low-power mode to schedule the next highest priority transaction, if all the other cores are busy executing higher priority

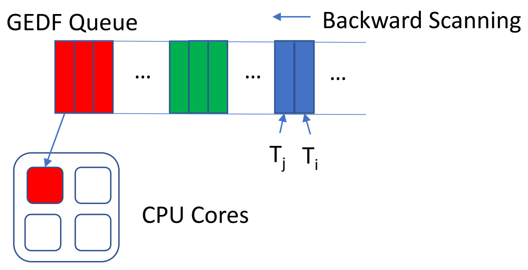


Fig. 5 Real-Time Query Aggregation in the GEDF Queue for User Transactions

transactions. If multiple cores are idle in such a case, it *wakes up the idle core in the shallowest low-power mode* to schedule the transaction, because it usually takes more time and energy to wake up a core in a deeper low-power mode. A tie is broken arbitrarily.

Ideally, it is desirable to process real-time transactions in a timely fashion using the minimum number of cores rather than waking up a core in a low-power mode; however, this is a challenging problem beyond the scope of this paper. A thorough investigation of workload consolidation in RTDBs is reserved for future work as discussed in Section 6.

Our approaches for real-time query aggregation and power management designed to reduce deadline misses and power usage are discussed in detail in the following subsections.

4.2 Aggregating Real-Time Queries

In this paper, we only merge read operations of multiple user transactions. We do not merge update transactions, because each update transaction in an RTDB periodically updates a specific temporal data object in a dedicated manner to maintain the freshness [Lam and Kuo(2006)].

When a user transaction T_i is inserted into the GEDF queue, Algorithm 2 is executed for real-time query aggregation. First, R_i and R_j of T_i and T_j where $j = i - 1$ in the GEDF queue are identified as shown in Figure 5. If $|R_i \cap R_j| \geq \theta$ where θ is the specified threshold, the common data in their read sets are: $R_{ij} = R_i \cap R_j$. In general, a query optimizer in a database system analyzes queries' data accesses for performance optimization. Thus, we exploit the read set information provided by the query optimizer for real-time query aggregation, incurring little additional overhead. Alternatively, real-time transactions are often canned, i.e., predefined, to access and process specific data elements to enhance the timeliness [Lam and Kuo(2006)]. In such a case, R_i and R_j are known *a priori*. In both cases, we express R_i and R_j as bit strings of length μ that is the maximum transaction size in terms of the total number

Algorithm 2: RTQA(T_i) /* Real-Time Query Aggregation */

```
input : $T_i$ ($i^{th}$ user transaction in the GEDF queue)
1  $j = i - 1$;
2  $cnt = 0$;
3  while $j \geq 0$ and $cnt < MaxScan$ do
4      $R_i$ = Read Set($T_i$);
5      $R_j$ = Read Set($T_j$);
6      if $|R_i \cap R_j| \geq \theta$ then
7          $R_{ij} = R_i \cap R_j$;
8          if $R_j$ is merged already then
9              return;
10         else
11             $i = j$;
12             $j - -$;
13     else
14         $j - -$;
15     $cnt$++;
```

of data accessed by an arbitrary transaction in the RTDB.[4] We compute R_{ij} by doing a bitwise AND operation between R_i and R_j, which is an O(1) time operation.

If T_i and T_j are merged because $|R_i \cap R_j| \geq \theta$ as specified in Line 6 of Algorithm 2, we check whether R_j has already been merged with the real-time queries ahead of T_j in the GEDF queue in Line 8. For example, T_j in Figure 5 may have already been aggregated with the green queries in the GEDF queue. If this is true, no more aggregation is needed. Hence, the algorithm returns in Line 9. Otherwise, the loop is iterated to further aggregate the user transactions ahead of T_i (if any) for at most $MaxScan$ times where $MaxScan$ is a pre-defined constant to bound the overhead for real-time query aggregation (Lines 10−12 in Algorithm 2).

On the other hand, if T_i and T_j cannot be merged (i.e., $|R_i \cap R_j| < \theta$), T_i is compared to the transactions in front of T_j for $MaxScan$ times (Lines 13 and 14). When the user transaction at the head of the GEDF queue is executed, the data read and processed by the transaction are shared by the other transactions, e.g., driving route requests or surveillance queries, with later deadlines that need to access and analyze common data, e.g., traffic data or surveillance images. When a later transaction runs, it uses the common data previously accessed and processed by an earlier deadline transaction as long as they are still fresh. This is another reason to bound the GEDF queue scanning for real-time query aggregation by $MaxScan$. Entire GEDF queue scanning incurs large overhead. Also, data accessed by earlier transactions may become stale.

Notably, we do not actually rewrite or translate multiple real-time queries into a single query, but simply compute the intersection of their read sets

[4] If a certain data item is accessed by T_i, the corresponding bit in R_i is set to 1.

and execute the group of aggregated queries, e.g., the red queries in Figure 5, in the same core in the EDF order to minimize repeated data retrievals and analyses as discussed before. We take this approach to avoid the overhead for rewriting queries. Also, it is unclear how to assign a single deadline to a real-time query that actually combines multiple real-time queries with originally different deadlines into one, avoiding priority inversion.

4.3 Dynamic Power Management

A core in a multicore RTDB may become idle when the workload is relatively low. For example, there could be few user transactions/queries during the off-peak time in a transportation management system. Also, a core may become idle due to our real-time query aggregation method. Thus, in our approach, a core can enter a low-power mode for power saving when it becomes idle. In this section, a description of our approach for RTDB power saving is given.

Algorithm 3: RTDB Dynamic Power Management in an Idle Core

```
while true do
    if busy at time $t$ then
        Process transactions at the highest speed;
    else
        $\eta(t)$ = release time of the next earliest update $- t$;
        $\psi'(i)$ = estimated length of the $i^{th}$ idle interval;
        $\ell(i) = min(\eta(t), \psi'(i))$;
        for $j = 1; j < N_{cs}; j{+}{+}$ do
            if $\kappa\delta_j > \ell(i)$ then
                $j - -$;
                break;
        if $0 < j < N_{cs}$ then
            Switch to the $C_j$ state;
            while idle do
                Stay in the $C_j$ state;
        else if $j == 0$ then
            while idle do
                Use the lowest voltage and frequency /*highest P state*/;
    Compute $\psi'(i+1)$;
    Switch to the P0 state if this core is in the shallowest C state;
```

In Algorithm 3, our power saving method used by PM (Figure 2) for each core, in which N_{cs} stands for the number of the C-states, is summarized. In our approach, the RTDB runs in the P0 state using the highest frequency and voltage, if there is any update or user transaction to execute. In this way, the RTDB processes update and user transactions as fast as possible.

If the RTDB with N temporal data has no update or user transaction to execute at time t, our DPM scheme computes the remaining time to the next temporal data update, $\eta(t) = min_{k=1}^{N}(r_k - t)$, where r_k is the release time of the next periodic instance of the update task k that periodically updates temporal data object O_k. Thus, $\eta(t)$ is found in O(N).

Let us assume that the i^{th} idle interval since the RTDB system initialization, due to no user transaction arrival, begins at time t. For DPM, we estimate the expected length of the i^{th} idle interval at t. To this end, we use an exponentially weighted moving average (EWMA), which is effective to smooth out short-term fluctuations of the trend in a time series [Arce(2005)] (e.g., the lengths of idle intervals observed in time) and subject to much less overhead than machine learning techniques (e.g., [Zhang et al(2015)Zhang, Liu, Zhuang, Liu, Zhao, and Li]) are.

Specifically, let us define $\psi'(i-1)$ and $\psi(i-1)$ as the length of the $(i-1)^{th}$ idle interval *estimated* at the beginning of the $(i-1)^{th}$ interval for DPM and the actual length *measured* when the $(i-1)^{th}$ idle interval ended because of a user transaction arrival, respectively. Given them, at the beginning of the i^{th} idle interval, we estimate its length by taking an EWMA based on the history:

$$\psi'(i) = a \times \psi'(i-1) + (1-a) \times \psi(i-1) \tag{2}$$

where $\psi'(1)$ is initialized as a small value less than B_1, i.e., the break-even time of the shallowest low-power idle state, when the RTDB system starts to run. ($\psi(1)$ is measured when the first idle interval ends due to the first arrival of a user transaction since the system initialization). In Eq 2, a is the forgetting factor ($0 \le a \le 1$). For example, a DBA can set $a = 0.6$ to ensure that the impact of $\psi(i-1)$ is 1% on the smoothed value after 5 idle intervals by recursively solving Eq 2.

The expected length of the i^{th} idle interval, which will be used for RTDB power saving via DPM, is then:

$$\ell(i) = min(\eta(t), \psi'(i)) \tag{3}$$

If the RTDB has no update or user transaction to execute at time t, Algorithm 3 finds $\max\limits_{1<j<N_{cs}} \{\kappa\delta_j \le \ell(i)\}$ where κ (> 1) is a headroom constant used to compensate for possible errors in estimating $\psi'(i)$ and the overhead of executing Algorithm 3.[5] Each idle core then switches to the C_j state that is the estimated deepest low-power state to save power without increasing deadline misses. Since there are a constant number of the C states in a processor, the selection of C_j takes O(1) time.

When the transition to the C_j state completes at time t_b ($> t$), the actual idle interval begins. A core stays in the C_j state as long as it is idle. If TS picks the idle core to process an imminent periodic update job release or a new user transaction arrival at time $t_f (\ge t_b)$, the idle core transitions back to the P0

[5] Generally, a large κ value provides a lower miss ratio for saving less power or vice versa. In Section 5, $\kappa = 1.5$.

state. Thus, the *actual* length of the i^{th} idle interval that excludes any state transition latency is: $\psi(i) = t_f - t_b$. Algorithm 3 derives $\psi'(i+1)$ using Eq 2 based on $\psi'(i)$ and $\psi(i)$. Using $\psi'(i+1)$, Algorithm 3 is re-executed when a core becomes idle again in the future.

Notably, we compute Eq 2 for all incoming user transactions instead of a subset of them assigned to a specific core. Also, we consider $\eta(t)$ in Eq 3, although the release time of the next update to be assigned to a specific core could be longer than $\eta(t)$ if $m > 1$. We take this approach, because 1) it is hard to predict which user or update transaction will run in which core under GEDF; 2) overly optimistic idle interval length estimates may result in many deadline misses and power inefficiency when the actual idle interval lengths are shorter than the break-even times of the selected low-power modes; and 3) multiple cores may become idle simultaneously, in which considering all user transactions rather than individual ones assigned to a specific core may lead to fewer/smaller overestimates. Thus, our approach *saves power when the system is underutilized, while reducing the miss ratio for higher workloads* via careful real-time transaction scheduling, query aggregation, and DPM.

However, if PM observes that the actual length of the i^{th} idle interval, $\psi(i)$, turns out to be shorter than the transition latency of C_j, δ_j, we consider that the previous state transition to C_j has been ineffective. Thus, we compute the normalized estimation error with respect to δ_j as follows:

$$e(i) = \begin{cases} (\delta_j - \psi(i))/\delta_j & \text{if } \psi(i) < \delta_j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

To measure the estimation accuracy, we define the estimation error ratio:

$$P_e = 100 \times N_e/N_{all}\ (\%) \tag{5}$$

where N_e and N_{all} represent the total number of the occurred estimation errors and that of all the transitions to low-power states, respectively.

Also, we measure the average size of the normalized estimation errors:

$$M_e = 100 \times \sum_{i=1}^{N_e} e(i)/N_e\ (\%) \tag{6}$$

Note that no estimation error occurs due to periodic updates, because the periods of temporal data updates in the RTDB are known *a priori*. In Section 5, Table 1, Eq 5, and Eq 6 are used for performance evaluation.

On the other hand, if the idle interval is too short for a core to switch to any C_j state where $j \geq 1$, it switches to the highest P state with the lowest voltage and frequency and stays in that state until it has to switch back to the P0 state to process a user or update transaction (Lines 16−18). This method also with O(1) time complexity is viable, since the DVFS latency is tens of microseconds only [Mazouz et al(2014)Mazouz, Laurent, Pradelle, and Jalby]. Hence, we do not consider any estimation error in this case.

The time complexity of executing Algorithm 3 for m cores is O(mN) where m is a constant in an RTDB system. Thus, the total time complexity of our approach, which consists of the real-time query aggregation and DPM techniques described in Algorithms $1-3$, is O(N). It is linear in terms of the number of the temporal data in the RTDB but independent of the number of real-time queries. Thus, our approach is applicable to different RTDB applications with various user transaction arrival rates and data access patterns.

5 Performance Evaluation

In this section, the performance of our approach and baseline is compared thoroughly. In Section 5.1, a description of the simulation set-up and baselines is given. In Section 5.2 the performance evaluation results are discussed. Also, the accuracy of estimating idle interval lengths is analyzed in Section 5.3.

5.1 Simulation Set-Up and Baselines

In this subsection, our simulation models and baselines are described.

5.1.1 Uniprocessor RTDB Settings

Our simulation settings summarized in Tables 2 and 3 are similar to the ones used in other RTDB research modeled after data-intensive real-time applications, e.g., air traffic control, fire detection, and engine diagnosis [Xiong et al(2008)Xiong, Han, Lam, and Chen, Han et al(2014)Han, Chen, Xiong, Lam, Mok, and Ramamritham,Kang and Son(2012),Gustafsson et al(2005)Gustafsson, Hallqvist, and Hansson].

Our simulation settings for temporal data updates are summarized in Table 2. As shown in the table, there are 1000 temporal data objects in our (simulated) uniprocessor RTDB ($m = 1$). Each data object O_i ($1 \leq i \leq 1000$) is periodically updated by an update stream $Stream_i$ associated with an estimated execution time EET_i and an update period P_i. EET_i is uniformly distributed in a range [3ms, 6ms]. When a periodic update job is generated, the actual update execution time AET_i is derived by applying a normal distribution $Normal(EET_i, \sqrt{EET_i})$ to $Stream_i$ to model potential variations in update execution times.

For $m = 1$, the total update workload, W_u, requires approximately 50% CPU utilization. Also, higher priority is given to updates to maintain the data freshness. Thus, all deadlines of update transactions are met. In the rest of this paper, we only consider the miss ratio of user transactions.

The total load applied to the RTDB is W_u ($\approx$ 50%) + user transaction load. In our uniprocessor experiments, $60\% - 160\%$ total loads are applied to evaluate the deadline miss ratio and power consumption of our approach and the baselines for different workloads.

Table 2 Simulation Settings for Data and Updates

Parameter	**Value**
m (number of cores)	1, 2, 4, 8
#Data Objects	1000, 2000, 4000, 8000 for m = 1, 2, 4, 8, respectively
Update Period	$Uniform[100ms, 50s]$
EET_i	$Uniform[3ms, 6ms]$
Update Load (W_u)	$\approx$ 50%, 100%, 200%, 400% for m = 1, 2, 4, 8, respectively

Table 3 Simulation Settings for User Transactions

Parameter	**Value**
EET_i	$Uniform[5ms, 20ms]$
Actual Exec. Time	$Normal[EET_i, \sqrt{EET_i}]$
N_{DATA_i}	$EET_i \times$ *Data Access Factor* = [5, 20]
#Actual Data Accesses	$Normal(N_{DATA_i}, \sqrt{N_{DATA_i}})$
Slack Factor	[10, 20]

Table 3 summarizes the simulation set-up for user transactions. In this paper, a source, $Source_i$, generates a series of real-time user transactions whose inter-arrival time is distributed exponentially. $Source_i$ is associated with EET_i. In this paper, $EET_i = Uniform[5ms, 20ms]$. Using multiple sources, we statistically generate transaction groups with different average execution times and numbers of data accesses. To increase the workload applied to the RTDB, we increase the number of sources. As a result, more user transactions arrive per unit time. When a user transaction is generated, the actual execution time AET_i is generated by applying the normal distribution $Normal(EET_i, \sqrt{EET_i})$ to introduce execution time variations in a series of user transactions produced by $Source_i$.

We derive the average number of data accesses for $Source_i$ in proportion to EET_i; that is, $N_{DATA_i} = data\ access\ factor \times EET_i = [5, 20]$ as summarized in Table 3. Thus, a longer transaction generally accesses (and processes) more data. When generating a user transaction, the actual number of data for the transactions generated by $Source_i$ to access and process is varied by applying $Normal(N_{DATA_i}, \sqrt{N_{DATA_i}})$.

For a user transaction, *deadline = arrival time + estimated execution time* $\times$ *slack factor* in this paper. A slack factor is uniformly distributed in a range (10, 20). For an update transaction, however, *deadline = next update period.*

5.1.2 Multiprocessor RTDB Settings

Related work on multiprocessor RTDB is relatively scarce [Li et al(2011)Li, Chen, Xiong, and Li,Kang and Chung(2015),Kang and Chung(2017)]. In this paper, we evaluate the performance of our approach and the baselines using the simulation parameters in Tables 2 and 3 for $m = 2, 4$, and 8 as well. For $m = 2$, we consider 2,000 temporal data objects as shown in Table 2 to approximately double the update workload to 100%, while generating more user transaction loads. Thus, the total load applied to the RTDB is 100% + user transaction load. Specifically, 150%, 200%, 250%, 300%, and 350% ($1.75m$) total workloads are applied to evaluate the miss ratio and power consumption.

For $m = 4$, we consider 4,000 temporal data objects to increase the update workload to 200% (approximately). Hence, the total workload applied to the RTDB is 200% + user transaction load. Especially, 250%, 300%, 350%, 400%, 450%, 500%, 550%, 600%, 650%, and 700% ($1.75m$) workloads are applied to evaluate the miss ratio and power consumption.

For $m = 8$, we use 8,000 temporal data objects to increase W_u to 400%. Hence, the total workload applied to the RTDB is 400% + user transaction load. Specifically, 600%, 700%, 800%, 900%, 1000%, 1100%, 1200%, 1300%, and 1400% ($1.75m$) workloads are applied to evaluate the miss ratio and power consumption.

5.1.3 Baselines

In this paper, we simulate the RTDB system architecture depicted in Figure 2. For performance evaluation, we consider a baseline, called the *Power-Unaware RTDB (PU-RTDB).* In PU-RTDB, GEDF scheduling and 2PL-HP are supported to process real-time transactions as discussed in Section 3. Thus, the baseline and our approach apply the same scheduling and concurrency control techniques for fair performance comparisons. However, no query aggregation or power saving is considered in PU-RTDB, similar to most existing RTDBs. Thus, PU-RTDB represents state-of-the-art RTDBs. In contrast, our approach, called Query Aggregation (QA), supports real-time query aggregation and DPM.

In fact, we have directly applied a well-known query aggregation technique effective for non-RTDBs [Lang and Patel(2009)] as another baseline. However, we have observed more than 90% of user transactions miss their deadlines even when the total workload is only 60% ($m = 1$) and user transactions are delayed only until the length of the EDF queue becomes 5. This result illustrates the need for real-time query aggregation in RTDBs. In the rest of this paper, we compare the performance of our approach to that of PU-RTDB without considering the query aggregation scheme for non-RTDBs [Lang and Patel(2009)] any further.

In this paper, we omit the performance results of QA with the DVFS option (Lines 16−18 in Algorithm 3) and let the idle core remain in the C0 state, if the estimated idle interval is shorter than B_1 (the break-even time for C1). This is because the DVFS method achieves less than 2% additional power saving in our experiments when we assume that a core using the lowest voltage and frequency consumes 0.75W, which is the average of the C0 and C1 power consumption in Table 1. We have observed that a core usually switches to the C1 state instead of doing DVFS, since the transition delay of C1 (0.1ms in Table 1) is negligible in terms of real-time transaction arrivals and executions. However, the DVFS option could be still useful when the system normally has short idle intervals, for example, during the peak time in transportation management. A thorough investigation is reserved for future work and discussed in Section 6.

In the rest of this paper, QAxx indicates QA with query aggregation probability of xx%. For example, two arbitrary queries can be merged into one query with 10% probability in QA10. We consider a broad spectrum of query aggregation probabilities, 10%−70%, for extensive performance evaluation, because data access patterns may vary from RTDB application to application. The data access pattern in one RTDB application may also vary based on the real-world status, e.g., traffic or weather conditions.

5.2 Performance Evaluation Results

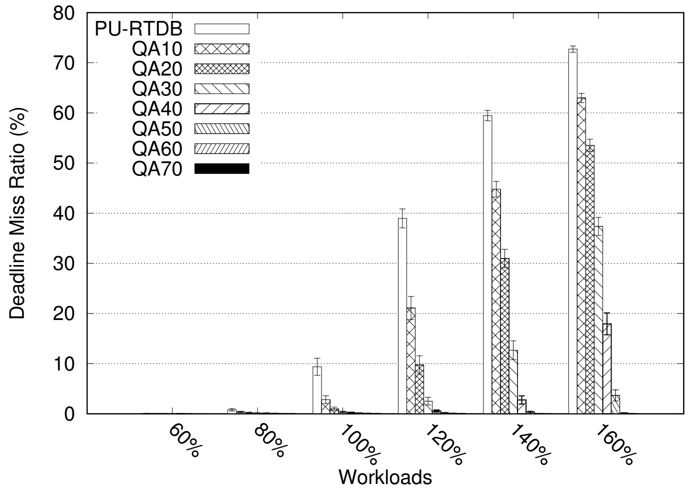
In this subsection, the deadline miss ratio and utilization of PU-RTDB and QA as well as the power saving achieved by QA for m = 1, 2, 4, and 8 are discussed.

5.2.1 Experiment Set 1: $m = 1$ and $W_u = 50\%$

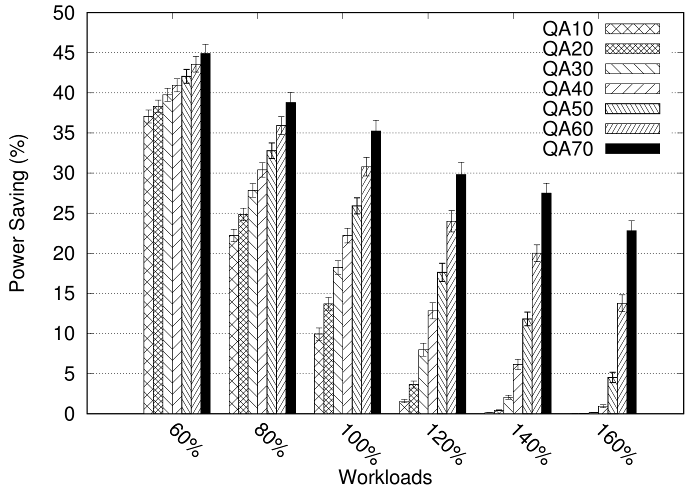
In Figure 6(a), the miss ratio of the PU-RTDB ranges between $0.33 \pm 0.3\% - 72.72 \pm 0.62\%$ as the load is increased from 60% to 160%. QA10's miss ratio ranges between $0.06 \pm 0.04 - 62.97 \pm 0.92\%$. Thus, compared to PU-RTDB, QA10 reduces the miss ratio by up to approximately 17% when 120% load is applied to the RTDB.

As shown in Figure 6(a), QA70 supports the lowest miss ratio among the tested approaches. Its miss ratio is near zero ($0.059 \pm 0.04\%$) even when the load is 160%; it decreases the miss ratio by roughly 72% compared to PU-RTDB. Although the miss ratio generally grows as the load increases, the growth rate is decreased substantially, if more real-time query aggregation is possible. Note that the miss ratio of the tested approaches is non-zero even when the total load is much below 100%, since some transactions may get aborted and restarted due to data/resource conflicts. This indicates the difficulty of processing real-time transactions in RTDBs.

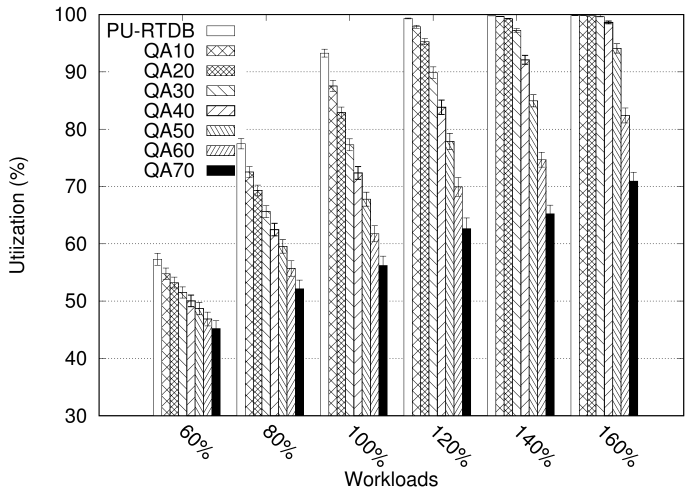
By aggregating real-time queries, QA reduces the workload and switches to a low-power mode when the RTDB becomes idle. In Figure 6(b), QA10 and QA70 decrease the total dynamic power consumption by up to 37% and 44% for the 60% load, respectively. In Figure 6(c), QA10 and QA70 reduce the utilization by up to approximately 6% and 37% compared to PU-RTDB for the 100% and 120% load, respectively. Notably, the magnitude of power saving by QA10−QA70 is bigger than that of the utilization decrease, because QA switches to a low-power state when the RTDB is idle. Being power-unaware, however, PU-RTDB cannot decrease the power consumption even when the system is underutilized. QA considerably decreases the power consumption especially when the load is relatively low as shown in Figure 6(b). The achieved power saving generally decreases as the load increases, because the RTDB



(a) Deadline Miss Ratio

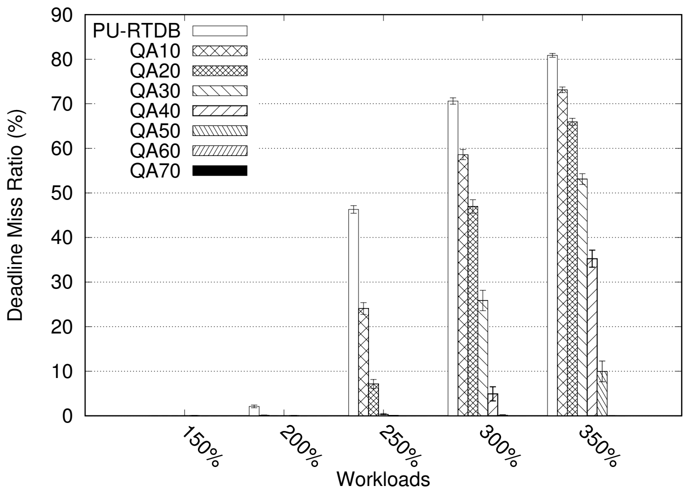


(b) Power Saving

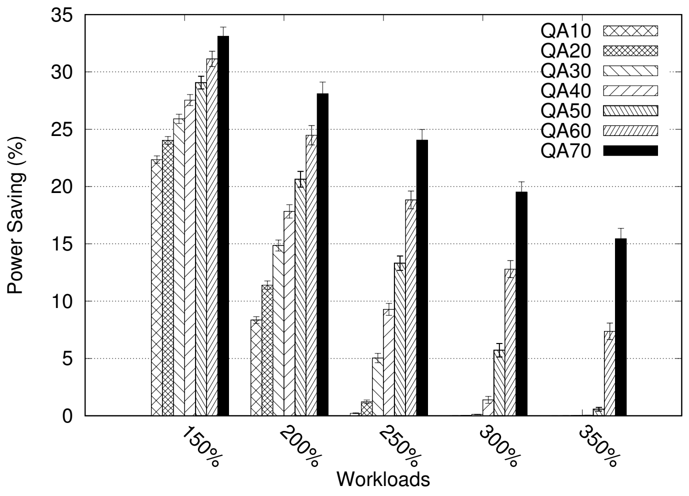


(c) Utilization

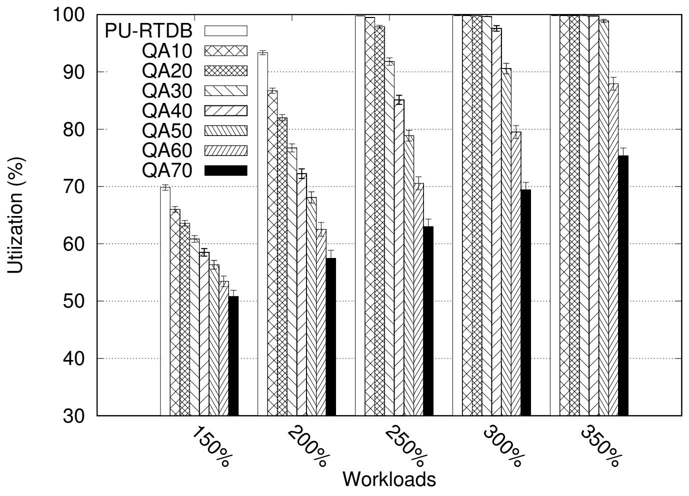
Fig. 6 Deadline Miss Ratio, Power Conservation, and Utilization for $m = 1$ and $W_u \approx 50\%$



(a) Deadline Miss Ratio

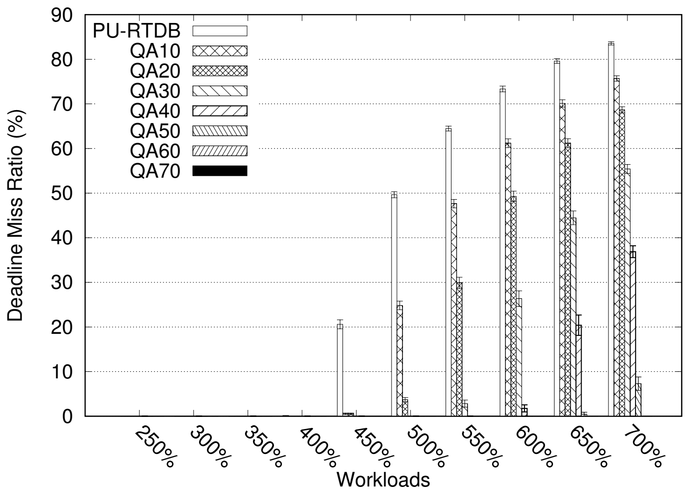


(b) Power Saving

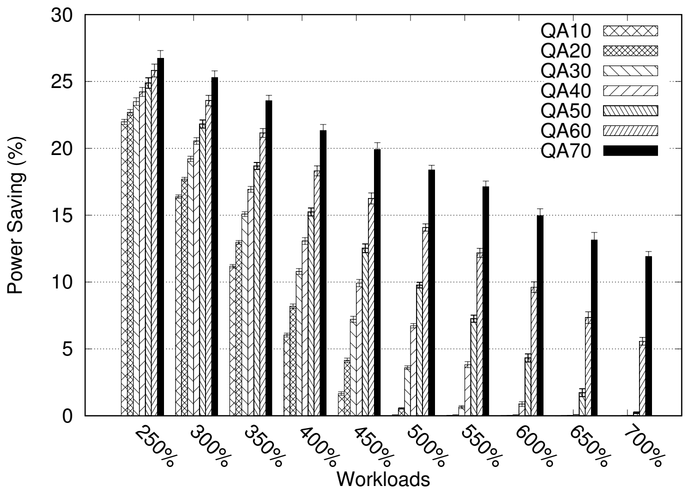


(c) Utilization

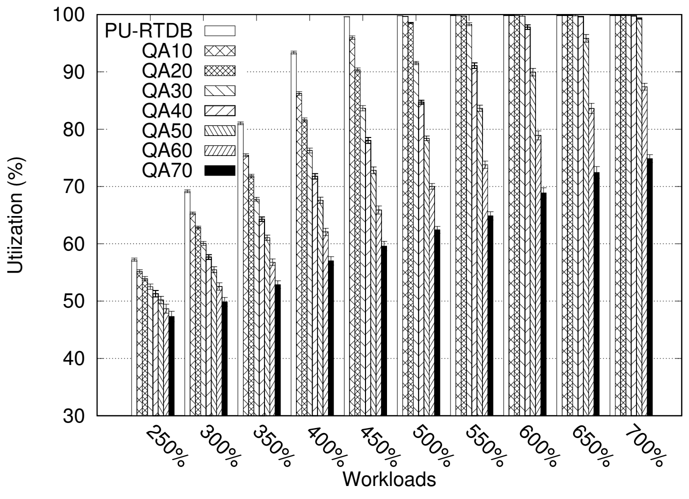
Fig. 7 Deadline Miss Ratio, Power Conservation, and Utilization for $m = 2$ and $W_u \approx 100\%$)



(a) Deadline Miss Ratio

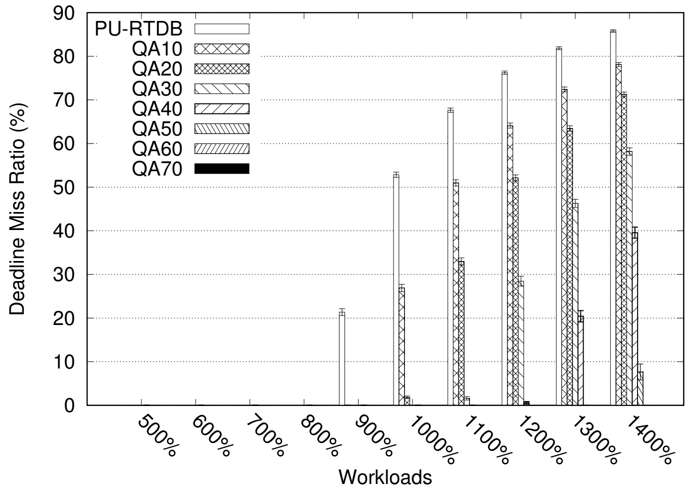


(b) Power Saving

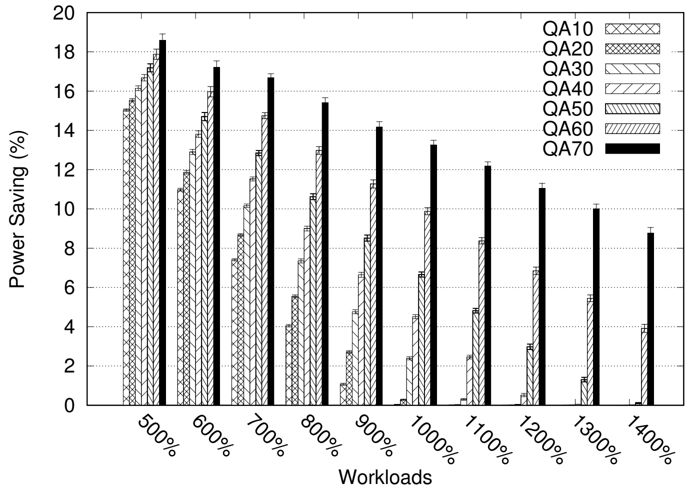


(c) Utilization

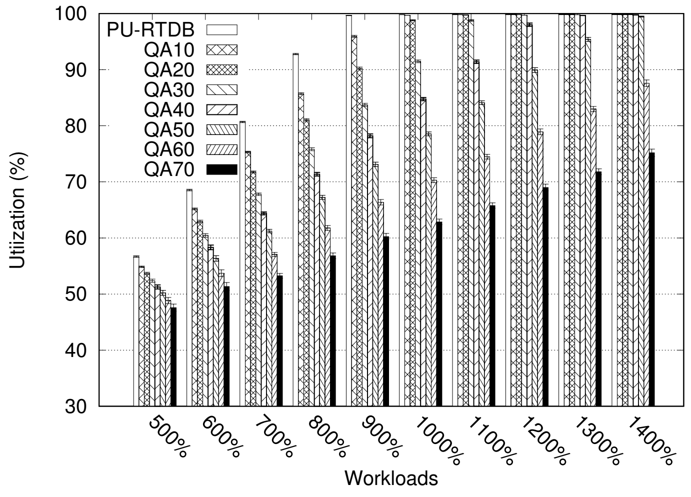
Fig. 8 Deadline Miss Ratio, Power Conservation, and Utilization for $m = 4$ and $W_u \approx 200\%$



(a) Deadline Miss Ratio



(b) Power Saving



(c) Utilization

Fig. 9 Deadline Miss Ratio, Power Conservation, and Utilization for $m = 8$ and $W_u \approx 400\%$

should run in the P0 state longer to process more real-time transactions as the load increases.

In summary, we observe that QA and DPM are effective in terms of managing the timeliness and power consumption in uniprocessor RTDBs.

5.2.2 Experiment Set 2: $m = 2$ *and* $W_u \approx 100\%$

Figure 7(a) shows the miss ratio of the tested approaches when the total workload is increased from 150% to 350% for $m = 2$ and update workload $\approx 100\%$. The miss ratio of PU-RTDB increases to $80.86 \pm 0.42\%$ as the load is increased to 350%. Compared to PU-RTDB, QA10 and QA70 reduce the miss ratio by up to 22% and 80% for 250% and 350% load, respectively.

As shown in Figure 7(b), QA10 and QA70 save power by more than 22% and 33% compared to PU-RTDB for the 150% load. The power conservation and miss ratio of QA70 for the 350% load are still more than 15% and near zero as shown in Figure 7(b) and Figure 7(a), respectively. QA70 is most effective in terms of miss ratio and power saving, because QA reduces more utilization and saves more power via DPM, if the probability of query aggregation is higher as shown in Figure 7(c).

From these results, we observe that QA is effective for $m = 2$ as well. It saves power when the system is underutilized (Figure 7(b)), while decreasing the miss ratio as the workload is increased (Figure 7(a)). We observe performance patterns similar to these results for $m = 4$ and $m = 8$ as follows.

5.2.3 Experiment Set 3: $m = 4$ *and* $W_u \approx 200\%$

In Figure 8(a), the miss ratio of every approach is near zero for $250\% - 400\%$ loads. For the 700% ($1.75m$) load, however, PU-RTDB misses more than 83% of the deadlines. Compared to PU-RTDB, QA10 decreases the miss ratio by up to approximately 24 % when the load is 500%. Further, QA70 reduces the miss ratio by more than 83% for 700% load. As shown in Figure 8(b), QA10 and QA70 save power by more than 21% and 26%, respectively, when the load is 250%. When the load is 700%, QA70 still saves power by more than 11%, while supporting the near zero miss ratio as shown in Figure 8(a). Similar to the previous experiments, QA-FA reduces the utilization compared to PU-RTDB as shown in Figure 8(c). As a result, they save power when the system is underutilized, while decreasing the miss ratio for higher workloads.

5.2.4 Experiment Set 4: $m = 8$ *and* $W_u \approx 400\%$

In Figure 9(a), the miss ratio of PU-RTDB increases fast as the workload is increased. Its miss ratio is over 85% for the 1400% load. Compared to PU-RTDB, QA10 reduces the miss ratio by up to approximately 26% when the load is 1000%. QA70 reduces the miss ratio by up to roughly 85% when the workload is 1400%. In Figure 9(b), QA10 and QA70 save power by up to more than 15% and 18%, respectively, when the load is 500%. In general, QA also reduces the utilization as shown in Figure 9(c).

Overall, our approach decreases the workload by aggregating real-time queries in the EDF and GEDF queue for $m = 1$ and $m > 1$, respectively. Also, it saves power by switching to an appropriate low-power mode when each core becomes idle. In summary, our approach effectively alleviates the tension between the competing requirements for the user transaction timeliness and power conservation in RTDBs. As a result, it considerably reduces the miss ratio and power consumption, while supporting the data freshness requirements in RTDBs.

5.3 Idle Interval Length Estimation Errors

Table 4 Estimation Error Rate and Magnitude

Exp. Set	Error Rate (P_e)	Error Size (M_e)
1	$0.21 \pm 0.02\% - 0.78 \pm 0.71\%$	$0.63 \pm 0.09\% - 6.35 \pm 3.71\%$
2	$0.0004 \pm 0.0001\% - 2.16 \pm 0.17\%$	$0.081 \pm 0.013\% - 1.23 \pm 0.54\%$
3	$0.014 \pm 0.001\% - 4.01 \pm 0.18\%$	$0.013 \pm 0.002\% - 2.16 \pm 1.03\%$
4	$0.001 \pm 0.0003 - 5.27 \pm 0.15\%$	$0.007 \pm 0.001 - 2.45 \pm 2.53\%$

We observe that the accuracy of our approach in terms of estimating the next idle interval length is acceptable as summarized in Table 4. In Experiment Set 1, the estimation error ratio P_e (Eq 5) ranges between $0.21 \pm 0.02\% - 0.78 \pm 0.71\%$. Our estimation accuracy is high in terms of P_e because: 1) the EWMA is effective to track the trend in a time series [Arce(2005)] and 2) the periods of temporal data updates used together with $\psi'(i)$ in Eq 3 are known in advance. Although M_e (Eq 6) ranges between $0.63 \pm 0.09\% - 6.35 \pm 3.71\%$, it has little impact on the miss ratio and power consumption because: 1) P_e is low, 2) a 6.35% estimation error is much smaller than the user transaction execution times and relative deadlines, and 3) the CPU spends only a small amount of time (less than 2% of the time) in the C3 state due to δ_3 that is 100 and 5 times longer than δ_1 and δ_2, respectively. In this paper, we observe that the amount of the time spent by QA in the C-states generally decreases in C0, C2, C1, and C3 order.

In Table 4, our approach shows acceptable M_e and P_e for $m = 2, 4$, and 8 as well. From the table, we observe that M_e is significantly reduced compared to that of Experiment Set 1 where $m = 1$, but the highest P_e in Experiment Sets $2 - 4$ is higher than that measured in Experiment Set 1. In our experiments, as m is increased, higher workloads are used for performance evaluation as discussed before. As a result, the inter-arrival time between user transactions may decrease and idle interval lengths could be overestimated more often for a bigger m, potentially increasing P_e (Eq 5). However, if the actual length of the $(i-1)^{th}$ idle interval, $\psi(i-1)$ in Eq 2, becomes shorter due to the decreased inter-arrival time, M_e (Eq 6) may decrease. This is because $\psi'(i)$ in Eq 2 decreases if $\psi(i-1)$ becomes shorter. Consequently, $\ell(i)$ in Eq 3 used for DPM could be decreased, possibly reducing $e(i)$ in Eq 4 and the average size of the normalized estimation errors, M_e.

6 Open Research Issues

Research on power-aware RTDBs has received relatively little attention despite the importance. There are several relevant research directions including the ones outlined in the following:

- Improving the accuracy of idle interval length estimation: Enhancing the accuracy in estimating the next idle interval is a key issue for DPM in RTDBs. Although machine learning techniques could be applied to enhance the accuracy, they are relatively heavy in terms of computation. Moreover, a supervised learning method may perform poorly when the training set used to develop a model does not represent unforeseen idle interval lengths, which may vary in time, well. On the other hand, an unsupervised learning algorithm may take a long time or even fail to derive (learn) unknown parameters necessary for a prediction. In general, the applicability of machine learning to timeliness and power management in RTDBs is largely unknown. Alternatively, feedback control techniques [Phillips and Nagle(1995)] could be applied to manage estimation errors by adapting the RTDB behavior in the closed-loop system.
- Integrating race-to-idle and never-idle methods: In this paper, we mainly consider the race-to-idle approach.[6] On the other hand, in [Kang and Chung(2015), Kang and Chung(2017)], only the never idle method is considered in embedded RTDBs. However, in certain RTDB applications, e.g., transportation management, workloads could be diurnal and/or seasonal. In such applications, the RTDB could apply a never-idle method, e.g., DVFS, to reduce deadline misses and power consumption when the load is usually high (e.g., rush hours), while applying a race-to-idle method during the off-peak time. Generally, this is a largely open problem.
- Transaction scheduling and concurrency control in multicore/many-core RTDBs: Although our real-time query aggregation scheme based on GEDF is effective for RTDBs running on a platform with a relatively small number of cores, it is unknown how to design power-aware RTDBs for many-core platforms with significant migration costs. Novel techniques are needed to address this problem. For example, it could be effective to partition certain update and user transactions together into clusters of cores such that user and update transactions that update the data accessed by the user transactions are scheduled in the same cluster, while supporting real-time query aggregation via global scheduling within a cluster. However,

[6] Our DVFS scheme, which switches to the lowest frequency when the idle interval is too short for the C1 state, saves power by less than 2% as discussed in Section 5.

this is challenging as partitioned scheduling is NP-hard and data access patterns may vary in time. Li et al. [Li et al(2011)Li, Chen, Xiong, and Li] investigated how to partition temporal data updates to $m \geq 2$ processors under EDF, maintaining the temporal consistency; however they do not consider power efficiency or data access patterns of user transactions. Moreover, multi-version concurrency control could be supported to allow user transactions to access an old version of temporal data in a controlled manner rather than waiting for temporal data updates. A challenge is how to avoid unbounded data staleness, while minimizing data/resource contention, which may adversely affect the data freshness and timeliness in RTDBs, respectively. Although transaction scheduling and concurrency control in RTDBs have been studied extensively [Lam and Kuo(2006)], related work on power-aware multiprocessor RTDBs is scarce.

– Workload Consolidation: In cloud computing, workloads can be consolidated and idle resources, e.g., idle CPU cores or servers, can be turned off to reduce energy consumption. However, care should be taken, because workload consolidation often incurs substantial overhead in terms of potential performance penalty and resource/energy wastage to turn on/off resources and migrate virtual machines [Nguyen et al(2015)Nguyen, Francesco, and Yla-Jaaski,Srikantaiah et al(2008)Srikantaiah, Kansal, and Zhao]. Although workload consolidation could be applicable to long-term, non-real-time data storage and analytics using abundant resources and time, RTDBs may miss many deadlines and waste power/energy if such techniques are directly applied. Essentially, the problem of consolidating real-time transactions requires the RTDB to minimize the number of the cores used to process real-time transactions in a timely manner, which can be reduced to a bin packing problem that is NP-hard. To aggravate the problem, it should be solved efficiently at runtime to consolidate real-time transactions with little overhead. Further, it could be hard to predict real-time user transaction workloads. For example, many real-time data service requests may arrive simultaneously upon unexpected traffic incidents or abnormal situations in a smart building. In such a case, the RTDB may miss many deadlines due to the overhead to turn on the cores previously turned off for power saving to assign or migrate real-time transactions to them. Also, energy could be wasted considerably, if the time interval between turning the cores off and on is not long enough. In this paper, we take a safer approach that aggregates real-time queries and schedules them together in one core to reuse the data accessed and processed by earlier deadline transactions to decrease the miss ratio, while opportunistically saving power via DPM. In sum, workload consolidation could have potential to further save power/energy in RTDBs; however, it is an open problem with significant challenges. An in-depth investigation is reserved for future work.

Although this list is neither exhaustive nor complete, a key lesson we learned from this work is that it is possible to reduce both deadline misses and power consumption in RTDBs with one or more processors, while meeting the data freshness requirements. This approach could be extended to further enhance the timeliness and power efficiency of RTDBs by exploring more effective real-time query optimization, transaction scheduling, concurrency control, and RTDB system design techniques that consider inherent RTDB characteristics, real-time data semantics, or advanced hardware/operating system features.

7 Conclusions

In data-intensive real-time embedded and cyber-physical system applications, e.g., traffic control and surveillance, it is desirable yet challenging to process real-time transactions in a timely manner using fresh data in real-time databases, while consuming less power. To address the challenge, we devise a cost-effective approach for real-time query aggregation and dynamic power management to reduce both deadline misses and power consumption in real-time databases rather than improving one of them by potentially degrading the other via trade-offs. Further, our approach does not require complex system modeling, tuning, or constrained real-time transaction models different from most existing work on power-aware real-time databases (as discussed in Section 2). The deadline miss ratio and power consumption of our approach are thoroughly compared to those of modern power-unaware real-time databases. Our approach decreases the deadline miss ratio and power consumption by up to approximately 86% and 45%. Despite the importance, relatively little work has been done on power-aware real-time databases. In the future, we will continue to explore open issues for power-efficient real-time data services including the ones discussed in Section 6 to promote further research.

Acknowledgment

We appreciate anonymous reviewers for their help to improve the paper. This work was supported, in part, by NSF grant CNS-1526932.

References

[Arce(2005)] Arce G (2005) Nonlinear Signal Processing: A Statistical Approach. Wiley

[Babu and Bizarro(2005)] Babu S, Bizarro P (2005) Adaptive Query Processing in the Looking Glass. In: Conference on Innovative Data Systems Research

[Bambagini et al(2016)Bambagini, Marinoni, Aydin, and Buttazzo] Bambagini M, Marinoni M, Aydin H, Buttazzo G (2016) Energy-Aware Scheduling for Real-Time Systems: A Survey. ACM Transactions on Embedded Computing Systems 15(1)

[Baruah et al(2014)Baruah, Bertogna, and Buttazzo] Baruah S, Bertogna M, Buttazzo G (2014) Multiprocessor scheduling for real-time systems. Springer

[Bastoni et al(2010)Bastoni, Brandenburg, and Anderson] Bastoni A, Brandenburg BB, Anderson JH (2010) An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In: IEEE Real-Time Systems Symposium

[Cao and Ravindran(2014)] Cao G, Ravindran AA (2014) Energy Efficient Soft Real-Time Computing through Cross-Layer Predictive Control. In: International Workshop on Feedback Computing

[Deshpande et al(2007)Deshpande, Ives, and Raman] Deshpande A, Ives Z, Raman V (2007) Adaptive Query Processing. Foundations and Trends in Databases 1(1):1–140

[Devi and Anderson(2008)] Devi U, Anderson J (2008) Tardiness Bounds under Global EDF Scheduling on a Multiprocessor. Real-Time Systems 38(2):133–189

[D'souza and Rajkumar(2017)] D'souza S, Rajkumar R (2017) Thermal Implications of Energy-Saving Schedulers. In: Euromicro Conference on Real-Time Systems

[Fu et al(2016)Fu, Calinescuy, Wang, Li, and Xue] Fu C, Calinescuy G, Wang K, Li M, Xue CJ (2016) Energy-Aware Real-Time Task Scheduling on Local and Shared Memory System. In: IEEE Real-Time Systems Symposium

[Guo et al(2017)Guo, Bhuiyan, Saifullah, Guan, and Xiong] Guo Z, Bhuiyan A, Saifullah A, Guan N, Xiong H (2017) Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks. In: Euromicro Conference on Real-Time Systems

[Gustafsson et al(2005)Gustafsson, Hallqvist, and Hansson] Gustafsson T, Hallqvist H, Hansson J (2005) A Similarity-Aware Multiversion Concurrency Control and Updating Algorithm for Up-To-Date Snapshots of Data. In: Euromicro Conference on Real-Time Systems

[Han et al(2014)Han, Chen, Xiong, Lam, Mok, and Ramamritham] Han S, Chen D, Xiong M, Lam KY, Mok AK, Ramamritham K (2014) Schedulability analysis of deferrable scheduling algorithms for maintaining real-time data freshness. IEEE Transactions on Computers 63(4):979 – 994

[Han et al(2016)Han, Lam, Chen, Xiong, Wang, Ramamritham, and Mok] Han S, Lam KY, Chen D, Xiong M, Wang J, Ramamritham K, Mok AK (2016) Online mode switch algorithms for maintaining data freshness in dynamic cyber-physicalsystems. IEEE Transactions on Knowledge and Data Engineering 28(3):756–769

[Hu and et al.(2015)] Hu S, et al (2015) Data Acquisition for Real-time Decision-making under Freshness Constraints. In: IEEE Real-Time Systems Symposium

[Imes et al(2015)Imes, Kim, Maggio, and Hoffmann] Imes C, Kim DHK, Maggio M, Hoffmann H (2015) POET: a portable approach to minimizing energy under soft real-time constraints. In: IEEE Real-Time and Embedded Technology and Applications Symposium

[Irani et al(2007)Irani, Shukla, and Gupta] Irani S, Shukla S, Gupta R (2007) Algorithms for power savings. ACM Transactions on Algorithms 3(4)

[Kang(2016)] Kang KD (2016) Reducing Deadline Misses and Power Consumption in Real-Time Databases. In: IEEE Real-Time Systems Symposium

[Kang and Chung(2015)] Kang W, Chung J (2015) QoS Management for Embedded Databases in Multicore-Based Embedded Systems. Mobile Information Systems

[Kang and Chung(2017)] Kang W, Chung J (2017) Energy-efficient response time management for embedded databases. Real-Time Systems 53(2):228–253

[Kang and Son(2012)] Kang W, Son SH (2012) Power- and time-aware buffer cache management for real-time embedded databases. Journal of Systems Architecture - Embedded Systems Design 58(6-7):233–246

[Kehr et al(2017)Kehr, Quinones, Langen, Boeddeker, and Schaefer] Kehr S, Quinones E, Langen D, Boeddeker B, Schaefer G (2017) Parcus: Energy-aware and Robust Parallelization of AUTOSAR Legacy Applications. In: IEEE Real-Time and Embedded Technology and Applications Symposium

[Kim et al(2016)Kim, Abdelzaher, Sha, Bar-Noy, and Hobbs] Kim JE, Abdelzaher T, Sha L, Bar-Noy A, Hobbs R (2016) Sporadic Decision-centric Data Scheduling with Normally-off Sensors. In: IEEE Real-Time Systems Symposium

[Kim et al(2017)Kim, Ward, Chisholm, Fu, Anderson, and Smith] Kim N, Ward BC, Chisholm M, Fu CY, Anderson JH, Smith FD (2017) Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning. Real-time Systems (Special Issue on Mixed-Criticality, Multi-Core, and Micro-Kernels)

[Kunjir et al(2012)Kunjir, Birwa, and Haritsa] Kunjir M, Birwa PK, Haritsa JR (2012) Peak power plays in database engines. In: International Conference on Extending Database Technology

[Lam and Kuo(2006)] Lam KY, Kuo TW (eds) (2006) Real-Time Database Systems. Kluwer Academic Publishers

[Lang and Patel(2009)] Lang W, Patel JM (2009) Towards eco-friendly database management systems. In: Biennial Conference on Innovative Database Systems Research

[Legout et al(2015)Legout, Jan, and Pautet] Legout V, Jan M, Pautet L (2015) Scheduling algorithms to reduce the static energy consumption of real-time systems. Real-Time Systems 51(2):153–191

[Li et al(2011)Li, Chen, Xiong, and Li] Li J, Chen JJ, Xiong M, Li G (2011) Workload-aware partitioning for maintaining temporal consistency on multiprocessor platforms. In: IEEE Real-Time Systems Symposium

[Madden et al(2005)Madden, Franklin, Hellerstein, and Hong] Madden SR, Franklin MJ, Hellerstein JM, Hong W (2005) TinyDB: An Acquisitional Query Processing System for Sensor Networks. ACM Transactions on Database Systems 30(1):122–173

[Mazouz et al(2014)Mazouz, Laurent, Pradelle, and Jalby] Mazouz A, Laurent A, Pradelle B, Jalby W (2014) Evaluation of CPU Frequency Transition Latency. Computer Science - Research and Development 29(3-4):187–195

[mcobject(2017)] mcobject (2017) eXtremeDB, a fast, reliable and cost-effective embedded database system for embedded systems and intelligent devices. URL `http://www.mcobject.com/emb`

[Nguyen et al(2015)Nguyen, Francesco, and Yla-Jaaski] Nguyen TH, Francesco MD, Yla-Jaaski A (2015) Virtual Machine Consolidation with Multiple Usage Prediction for Energy-Efficient Cloud Data Centers. In: IEEE International Conference on Cloud Computing (CLOUD)

[Phillips and Nagle(1995)] Phillips CL, Nagle HT (1995) Digital Control System Analysis and Design (3rd edition). Prentice Hall

[Ramamritham et al(2004)Ramamritham, Son, and DiPippo] Ramamritham K, Son SH, DiPippo L (2004) Real-Time Databases and Data Services. Real-Time Systems 28(2-3):179–215

[Srikantaiah et al(2008)Srikantaiah, Kansal, and Zhao] Srikantaiah S, Kansal A, Zhao F (2008) Energy aware consolidation for cloud computing. In: Workshop on Power Aware Computing and Systems (HotPower'08), USENIX Association

[Stankovic et al(1999)Stankovic, Son, and Hansson] Stankovic JA, Son SH, Hansson J (1999) Misconceptions About Real-Time Databases. IEEE Computer 32(6):29–36

[Tsiftes and Dunkels(2011)] Tsiftes N, Dunkels A (2011) A Database in Every Sensor. In: ACM Conference on Embedded Networked Sensor Systems

[Tu et al(2014)Tu, Wang, Zeng, and Xu] Tu YC, Wang X, Zeng B, Xu Z (2014) A System for Energy-Efficient Data Management. SIGMOD Record 43(1):21–26

[Valsan et al(2017)Valsan, Yun, and Farshchi] Valsan PK, Yun H, Farshchi F (2017) Addressing Isolation Challenges of Non-blocking Caches for Multicore Real-Time Systems. Real-time Systems (Special Issue on Mixed-Criticality, Multi-Core, and Micro-Kernels)

[Völp et al(2014)Völp, Hähnel, and Lackorzynski] Völp M, Hähnel M, Lackorzynski A (2014) Has energy surpassed timeliness? - Scheduling energy-constrained mixed-criticality systems. In: IEEE Real-Time and Embedded Technology and Applications Symposium

[Wires et al(2014)Wires, Ingram, Drudi, Harvey, and Warfield] Wires J, Ingram S, Drudi Z, Harvey NJA, Warfield A (2014) Characterizing storage workloads with counter stacks. In: USENIX Symposium on Operating Systems Design and Implementation

[Xiong et al(2008)Xiong, Han, Lam, and Chen] Xiong M, Han S, Lam KY, Chen D (2008) Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results. IEEE Transactions on Computers 57(7):952–964

[Xu et al(2013)Xu, Wang, and Tu] Xu Z, Wang X, Tu YC (2013) Power-Aware Throughput Control for Database Management Systems. In: International Conference on Autonomic Computing

[Xu et al(2015)Xu, Tu, and Wang] Xu Z, Tu YC, Wang X (2015) Online Energy Estimation of Relational Operations in Database Systems. IEEE Transactions on Computers 64(11):3223–3236

[Zhang et al(2015)Zhang, Liu, Zhuang, Liu, Zhao, and Li] Zhang Y, Liu Y, Zhuang L, Liu X, Zhao F, Li Q (2015) Accurate CPU Power Modeling for Multicore Smartphones. Tech. Rep. MSR-TR-2015-9, Microsoft