**Motivation**

Increasing amount of real-time data in embedded and cyber-physical systems

Desirable yet challenging to process real-time queries/transactions in a timely manner using fresh temporal data

- Dynamic workloads varying depending on the real-world status
- Data/resource contention → transaction aborts/restarts → deadline misses
- Timeliness, freshness, and power efficiency may compete with each other → Continuously strike a balance

**Traditional Real-Time Database**

Strict data temporal consistency model

- Temporal consistency of all temporal required
- Data importance or popularity not considered

No timing assurance

- Many deadlines misses given dynamic workloads or data/resource contention

Uniprocessor platforms

- Can't handle increasing data amounts & user queries

Power efficiency usually not considered

**Power-Aware Real-Time Embedded Database**

TS: Transaction Scheduler  
FM: Freshness Manager  
CC: Concurrency Controller

**Real-Time Query Aggregation**

GEDF Queue  
Backward Scanning

- Reduce both deadline misses and power consumption
  - Leverage data access skew (e.g., 80/20 access pattern) to combine queries accessing similar data
  - Merge similar queries into one by scanning the queue backwards to avoid affecting urgent queries, e.g., earliest deadline query

**Adaptive Data Freshness Management**

Adaptive data freshness management based on data popularity or importance

Data is hot if accessed at least as frequently as updated

Adapt update periods and validity intervals of cold data within a specified range

**Power Saving**

Reduce workloads via real-time query aggregation & data freshness adaptation

Transition to a low-power mode when the processor becomes idle

- Predict the idle interval length based on history
- Switch to an appropriate ACPI C state

**Adaptive Control**

Load sharing between cores via adaptive control theoretic technique

- P1(k), P2(k), ..., Pm(k): Load balancing probabilities in the kth control period
- F1(k), F2(k), ..., Fm(k): Clock frequencies for DVFS
- Adjust load sharing probabilities based on slack capacity of each core and data access patterns of the user transaction assigned to the core
- Continuously adapt in the closed-loop system
- Self-tuning based on formal adaptive control theory

Race-to-idle vs. never-idle

- Never-idle
  - Complex modeling
  - Diminishing return of DVFS
- Race-to-idle
  - Challenging to predict workloads for state transition/consolidation
  - Time and energy overhead for state transitions
- Hybrid approaches

**References**