Energy-Aware Real-Time Scheduling: Comparing EDF, Static EDF, and Cycle-Conserving EDF Algorithms

Project Type 1 and 3

Christy Mettry (458) and Justin Wenzel (558)
CPR E 458//558 Real-Time Systems, Fall 2024
Department of Electrical and Computer Engineering
Iowa State University

December 11, 2024

Abstract

Our project presents the design and evaluation of energy-efficient real-time scheduling algorithms in a simulated environment. The group focuses on the algorithms Earliest Deadline First (EDF), Static EDF, and Cycle-Conserving EDF (CC-EDF). EDF is already a well-known scheduling algorithm that prioritizes tasks according to their deadlines. Static EDF is one step up, incorporating frequency scaling to save energy under worst-case execution assumptions. CC-EDF extends this by dynamically adjusting the CPU frequency in response to real-time task completion behavior, further optimizing energy efficiency.

We designed our own custom Python-based simulator for the group to analyze each algorithm. This allowed for evaluating each algorithm's performance under different workloads: light, medium, and heavy. During execution, we monitored multiple metrics, including energy consumption, deadline miss rate, and CPU utilization. Simulation results show energy efficiency vs. real-time performance trade-offs and CC-EDF adaptation to dynamic environments. This initial work gives an overview of real-time scheduling and creates an opportunity for further extension with more complex algorithms in the future.

1 Introduction

Real-time systems are everywhere, from embedded systems in automotive industries to IoT devices in smart homes. These systems must efficiently schedule tasks to meet the strict application deadlines and often operate in power-constrained environments. EDF is a commonly used algorithm in systems to ensure deadlines are met. However, EDF assumes worst-case execution times for tasks and maximum CPU utilization and does not adapt to opportunities for energy savings when tasks are completed early.

This project addresses the limitations of EDF by implementing and evaluating two extensions to the original EDF algorithms: Static EDF and Cycle-Conserving EDF. Static EDF utilizes CPU frequency scaling to reduce power consumption while continuing to assume worst-case execution times. CC-EDF optimizes this approach by dynamically adjusting CPU frequency based on real-time workload and task completion behavior.

To investigate the EDF algorithms and variances, we have developed a Python-based simulator for evaluating real-time scheduling within a controlled environment. The simulator we produced is not to an industry standard but implements the algorithms for an educational purpose to understand the trade-offs of different EDF scheduling algorithm variances. The simulator allowed for testing different workload levels, CPU frequencies, and energy consumption levels to provide a deeper insight into the algorithms' energy efficiency, deadline adherence, and CPU utilization.

2 Project Objectives & Scope

2.1 System Model

This project addresses task scheduling problems for real-time systems, like embedded systems and IoT devices. Most often, these environments contain periodic tasks that require robust timing, such as collecting data from sensors or control systems. Energy efficiency is also an important issue for such systems because most of them operate either on batteries or in places where power saving is critical.

The conceptual model consists of:

- **Periodic Tasks:** Characterized by a period, computation time, and deadline.
- **CPU Frequency Levels:** Adjustable frequencies to optimize energy efficiency in real-time performance.
- **Workload Scenarios:** Simulated task loads representing different utilization levels (light, medium, heavy).

Our simulator emulates this environment, providing a sandbox for exploring scheduling behavior and testing algorithm performance.

2.2 Problem Statement

Real-time systems must balance the demands of meeting strict deadlines and optimizing energy efficiency, especially in power-constrained environments. While EDF guarantees the meeting of deadlines under worst-case execution times, it fails to ensure the optimization of energy consumption during execution. Static

EDF introduces a simple solution of frequency scaling for energy saving but lacks the flexibility for dynamic finishing time scenarios. CC-EDF aims to fill the gaps between the other algorithms by dynamically adjusting CPU frequency according to actual task execution.

2.3 Objectives and Scope

The objectives of this work are as follows:

- Implement EDF, Static EDF, and CC-EDF scheduling algorithms in a custom simulation environment.
- Assess the algorithms under different workloads using key energy consumption, deadline misses, and CPU utilization metrics.
- Investigate the adaptability that CC-EDF achieves by leveraging early task completions for energy efficiency.

The scope is to implement a flexible simulator that can easily be used to evaluate our current EDF algorithms. Although the simulator will not be an industry-verified tool, it will still provide an educational framework for exploring real-time scheduling algorithms and create a foundation that could be extended for other algorithms to study and evaluate performance.

3 Solution Methodology / Approach

Our approach is to develop a Python-based simulator that can simulate periodic task scheduling along with CPU frequency scaling. The tasks will be created to represent different attributes which include period, computation time, and deadline attributes. The simulator runs the tasks in a priority queue sorted by the task's current deadline and supports frequency adjustments for energy optimization.

3.1 Algorithms / Protocols / Architectures

Basic EDF Algorithm: The Basic EDF algorithm performs the task assignment based on absolute deadlines. At every time unit, a task with an earlier deadline is executed. In that duration, the processor will operate at the maximum frequency throughout to meet all deadlines. However, this approach is energy-inefficient as it does not account for other factors such as slack and frequency scaling.

Key Characteristics:

- Processor operates at maximum frequency, leading to high energy consumption.
- Tasks are executed based on WCET, which overestimates actual execution times.

- No consideration of slack or idle time for energy optimization.

Static EDF Algorithm: Static EDF improves Basic EDF by adding CPU frequency scaling that is based on the task sets utilization, defined as $U = \sum \text{WCET}/\text{period}$. The processor can then run at a fixed frequency, which is decided by the overall utilization. This allows for a potential save in energy during execution. However, execution times are still based on WCET, and the approach is limited by the static frequency adjustment.

Key Characteristics:

- Processor frequency is set based on utilization, balancing energy efficiency and task execution.
- Energy usage is reduced compared to Basic EDF, but execution times increase at lower frequencies.
- Assumes WCET.

Cycle-Conserving EDF Algorithm (CC-EDF): CC-EDF adjusts the CPU frequency dynamically by during runtime execution by using slack created from early execution. While Static EDF, uses the WCET, CC-EDF considers real task execution time, which is usually lower than WCET. By computing slack as a difference of time remaining to the earliest deadline and accumulate remaining execution times of all tasks to the next deadline, CC-EDF can scale up or down the frequency.Energy usage can be minimized while still ensuring tasks meet deadlines.

Key Characteristics:

- Dynamically calculates slack to perform frequency scaling.
- Adjusts processor frequency at each deadline.
- Accounts for actual execution times, often shorter than WCET.
- Ensures all deadlines are met by recalculating frequencies dynamically based on task deadlines and remaining workload.

3.2 Illustrative Example

To demonstrate the differences among EDF, Static EDF, and CC-EDF algorithms, we consider the following periodic task set:

- **Task1:** 10 ms of computation every 20 ms.
- **Task2:** 5 ms of computation every 15 ms.

System Setup: The task set has a total utilization of $U = 0.83$ $(10/20 + 5/15)$. The processor supports frequency scaling with the following configurations:

Frequency (GHz)	Power (W)
1.0	25
1.5	50
1.75	75
2.0	100

The simulation was run for 100 ms, and the key features of each algorithm are summarized below:

Basic EDF: Basic EDF operates at the maximum frequency (2.0 GHz) to meet all deadlines. Tasks are executed using their WCET:

- Task1 executes for 10 ms every 20 ms.
- Task2 executes for 5 ms every 15 ms.

This results in high energy consumption due to constant operation at peak power (100 W).

Static EDF: Static EDF uses a fixed frequency based on the task set utilization. Here, $U = 0.83$ requires the processor to operate at 1.74 GHz, as it is the lowest frequency that can support the workload. The Task execution times remain based on WCET, leading to energy usage similar to Basic EDF.

Cycle-Conserving EDF (CC-EDF): CC-EDF dynamically adjusts CPU frequency based on real-time task execution. Actual execution times are shorter than WCET, typically:

- Task1 completes in 7 ms instead of 10 ms.
- Task2 completes in 3.5 ms instead of 5 ms.

The algorithm calculates slack and reduces the frequency accordingly, significantly lowering energy usage while meeting all deadlines.

Simulation Results:

Algorithm	Energy (J)	Idle Time (ms)	Missed Deadlines
Basic EDF	8650	15	0
Static EDF	7500	0	0
CC-EDF	2010	29	0

Key Observations:

- **Basic EDF:** Consumes the highest energy (8650 J) due to constant operation at maximum frequency. Minimal idle time (15 ms) indicates over-utilization of resources.
- **Static EDF:** Eliminates idle time, reducing energy consumption to 7500 J. However, the lack of dynamic frequency scaling limits further savings.

- **CC-EDF:** Achieves the lowest energy consumption (2010 J) by dynamically scaling frequency based on real-time execution behavior. Increased idle time (29 ms) highlights its ability to conserve energy while meeting deadlines.

This illustrative example demonstrates how each algorithm is calculated and would be scheduled on a task set, it also shows how each algorithm might respond to such a task set with a higher utilization but our work will continue to explore other task set utilizations.

4 Implementation/Simulation Architecture

4.1 System Design and Key Components

The simulator is a discrete-event simulation framework to assess the performance of Basic EDF, Static EDF, and CC-EDF algorithms. This is implemented in Python as a modular and extensible architecture comprised of the following key components:

4.1.1 Scheduler Module

- Includes the scheduling logics of the three algorithms; it deals with task prioritization and respects the deadlines, whereas Static EDF and CC-EDF scale frequencies.
- A priority queue is maintained that ensures the tasks with the earliest deadlines are executed first.
- The EDFScheduler class manages the selection of the task to execute and monitors the deadlines. Its subclass CC EDFScheduler dynamically adapts the CPU frequency depending on the estimated slack and load.

4.1.2 Task Module

- This module represents periodic tasks. It maintains the WCET, actual execution time, period, and deadlines.
- Methods supported: the arrival of the task, executing the task, resetting the task, and monitoring task attributes over the simulation timeline.

4.1.3 Simulation Core

- Orchestrates the simulation by managing task arrivals, scheduling decisions, and execution tracking within a fixed time quantum.
- Integrates the scheduler and task modules to simulate workloads and logs relevant performance metrics, including energy consumption, idle time, and missed deadlines.

4.1.4 Energy Model

- Frequency-to-power mappings are defined within the configuration file, enabling the calculation of energy consumption for task execution, CPU frequency, and even idle periods.

4.1.5 Configuration Module

- All simulation parameters are centralized, including task definitions, CPU frequency levels, power consumption models, and execution time variability.
- This module allows for the modification of workload scenarios and energy models.

4.2 Workload Generation

Workload is generated through the config.py file which is also known as the configuration model that is used to define the simulation setup and task attributes. The user can set the simulator attributes through the provided GUI and it supports the generation of multiple workload scenarios. For CC-EDF, execution time variability is introduced into the simulator generation, the actual execution time are randomly selected within a range defined as a percentage of the WCET by the user. This allows the simulator to reflect real-world behavior of task completing early.

On top of being able to set task attributes users can set multiple other parameters also. The duration of the simulation can be set for ideal simulation execution times. The environment also offers the ability for a user to set the time quantum the system users for stepping, changing how fast the system can detect changes in tasks, enabling a fine-grained simulation of the tasks. The environment also supports a wide range of CPU frequencies and power levels the user may set.

4.3 Software Platforms and Tools

Developed in Python and leverages several libraries for its functionality:

4.3.1 Core Libraries

- `heapq`: For efficient management of the task priority queue.
- `random`: Used in CC-EDF to simulate variability in execution times.
- `copy`: Ensures the integrity of task data to ensure adjustments to tasks do not persist across simulations.

4.3.2 Graphical User Interface

The graphical user interface provides a simple interactive way for users to create task sets and change the simulation settings. The GUI provides schedule results in detail for all three algorithms for the user.

4.4 Outputs and Metrics

Multiple outputs are generated and documented by the simulator to monitor the performance of the algorithms:

- **Energy Consumption:** Total energy usage during the simulation, calculated based on execution time, frequency scaling, and idle power consumption.
- **Idle Time:** Total time the CPU spends idle during the simulation.
- **Missed Deadlines:** Counts the number of tasks that fail to meet their deadlines.
- **Task Execution Logs:** Includes detailed records of task arrivals, executions, and completions.

5 Evaluation

5.1 Test Plan and Setup

Assessing the performance of our simulator and the algorithms required a series of simulations to ensure reasonable results and proper execution. As a group, a total of fifteen different task sets were simulated, with each set consisting of different execution times and periods resulting in different utilizations. The task sets were then categorized based on their task set utilization, allowing us to test against low, medium, and high utilization sets outlined below:

- **Low Utilization:** 0-0.6
- **Medium Utilization:** 0.6-0.8
- **High Utilization:** 0.8-1.0

Using the fifteen task sets, we will evaluate multiple metrics outlined below:

- **Energy Consumption:** Total energy used in the simulated time
- **Idle Time:** The time the CPU was idle during the simulation
- **Missed Deadlines:** Tasks that did not complete before the deadline

5.1.1 Simulator Setup

The simulator allows a user to set multiple other metrics for a simulation. Our environment was set up as follows. A time quantum of 0.1 was used for a more fine-grained execution step to step. Simulation time of 1000 time units. Available scaling frequencies and energy units were (1.0:30), (1.5:45), (2.0:60), (2.5:90), and (3.0:120), with the format (frequency: power). The idle time for the CPU was set to 15 power units. Finally, for the CC-EDF task completion variability, the range was set for tasks to finish in the random range of 50%-80% of the original execution time.

5.2 Task Sets

Table 1, 2, and 3 provide a condensed overview of each task set that was tested in the evaluation of the simulation.

Table 1: Low Utilization Task Sets (0–0.6)

Set	**Util.**	**Tasks (Execution Time, Period)**
1	0.04	T1(1,50), T2(1,100), T3(1,200), T4(1,250)
2	0.20	T1(2,20), T2(3,60), T3(1,40), T4(2,100)
3	0.35	T1(4,20), T2(3,30), T3(1,40), T4(2,80)
4	0.45	T1(5,20), T2(3,30), T3(2,40), T4(4,80)
5	0.55	T1(6,20), T2(4,30), T3(3,40), T4(4,80)

Table 2: Medium Utilization Task Sets (0.6–0.8)

Set	**Util.**	**Tasks (Execution Time, Period)**
1	0.63	T1(10,30), T2(8,40), T3(6,60)
2	0.70	T1(12,40), T2(6,20), T3(4,40)
3	0.75	T1(9,30), T2(6,20), T3(9,60)
4	0.80	T1(10,50), T2(9,30), T3(12,40)
5	0.80	T1(15,50), T2(8,20), T3(6,60)

Table 3: High Utilization Task Sets (0.8–1.0)

Set	**Util.**	**Tasks (Execution Time, Period)**
1	0.85	T1(12,30), T2(10,40), T3(12,60)
2	0.90	T1(10,20), T2(12,30)
3	0.95	T1(15,30), T2(12,40), T3(9,60)
4	1.00	T1(12,30), T2(15,50), T3(14,70), T4(12,120)
5	1.00	T1(20,50), T2(15,30), T3(6,60)

5.3 Results

The results of the simulations are summarized in Figures 1 and 2. The figures illustrate the energy consumption and idle times for each algorithms for the

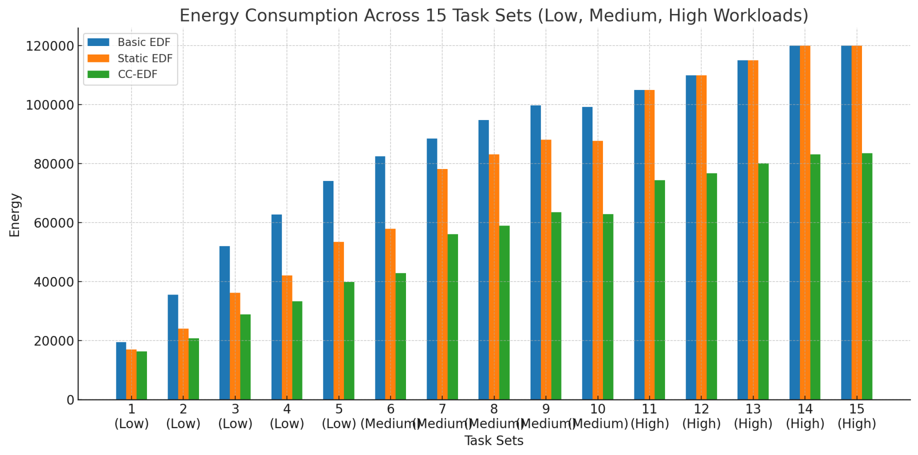

Figure 1: Energy Consumption

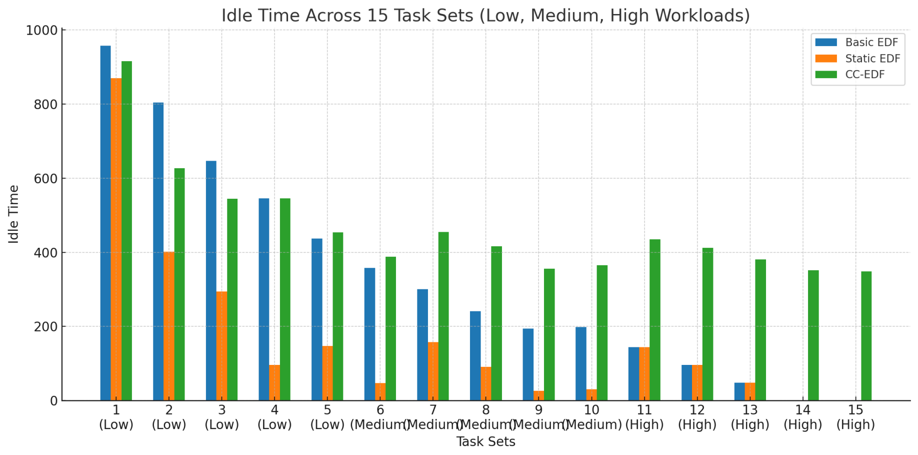

Figure 2: Enter Caption

fifteen different task sets evaluated

Energy Consumption: As shown in Figure 1, Basic EDF has the highest energy in all the workloads because it works at the maximum frequency for all the simulation times. Static EDF selects an appropriate fixed frequency depending on the utilization of the task set, thereby achieving high savings upon basic EDF. On the other hand, CC-EDF manages the best energy efficiency, especially on high utilizations, as the frequency scales down by the observed slack of tasks execution time.

Idle Time: Figure 2 shows the idle time of each algorithm. Static EDF has the minimum idle time since it closely approximates the processor frequency to the task workload, especially under medium and high utilizations. In contrast, Basic EDF, although all deadlines are met, leads to higher idle times since it cannot scale the frequency. CC-EDF strikes a balance: it can maintain energy efficiency while keeping the idle time within a reasonable range.

Importantly, no deadlines were missed by any of the algorithms for all the task sets at all the utilization levels. This guarantees the strength of all three schedulers for executing periodic tasks under real-time challenges. The overall outcome indicates that CC-EDF has the highest energy efficiency, and Static EDF minimizes the idle time, so both are preferable in power-critical systems over Basic EDF.

6 Conclusions

This project improved each of understandings of real-time scheduling algorithms by giving the opportunity to try deeper EDF algorithms. Each member on this team contributed to implementation, research, via it be the GUI implementation or algorithms. Each member contributed to the final document and presentation in work. Our final results proved that CC-EDF can improve energy efficiency while still maintaining no dead-lines being missed with a utilization under one. This project can be further continued to add more algorithms, and more in-depth features to replicate a true system and performance metrics but it lays a foundation for testing and algorithm understanding.

Self-Assessment of Project Completion

Project Learning Objectives	**Status**	**Pointers in Document**
Self-contained description of project goal, scope, and requirements	Fully Completed	Section 1, Pages 1-2; Section 2, Pages 2-3
Self-contained description of solutions (algorithms/protocol/applications/etc.)	Fully Completed	Section 3.1, Pages 3–4; Section 4, Pages 6–8

Adequate description of implementation details (data structures, pseudo code segments, libraries used, etc.)	Fully Completed	Section 4, Pages 6–8
Testing and evaluation –test cases, metrics, test results, any relevant performance results	Fully Completed	Section 5, Pages 8–11
Overall Project Success assessment	Mostly Successful	Section 6, Page 11

References

[1] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, Oct. 2001. DOI: `https://doi.org/10.1145/502059.502044`.

Team Members' Contributions (only for the 2-member team)

Tasks/Member	**Christy Mettry**	**Justin Wenzel**
Literature survey	Researched EDF and GUI libraries in Python	Researched Static EDF and CC-EDF with real-time scheduling
Design	Defined the GUI and task attributes for the simulator	Designed simulator structure and logic flow
Implementation	Implemented GUI and Basic EDF algorithm	Developed Static EDF and CC-EDF dynamic scaling algorithm
Testing/Evaluation	Ran usability tests and verified output results	Verified proper calculation of energy consumption, scaling behavior, and idle time occurrences
Preparation of the Report and Presentation	Drafted abstract and sections,1,2,5. The first half of the presentation slides	Drafted sections 3,4,5,6. The second half of presentation slides
Total percentage of contribution to the project	50%	50%