Real-Time Task Set Schedulability Analysis with Python

Introduction to Real-Time System: Homework 3

Hyungjun Shon

Dept. of Software and System Management Engineering, Sungkyunkwan Univ, shj1081@g.skku.edu

This report investigates the schedulability of real-time task sets using Python-based implementations of Rate Monotonic (RM), Deadline Monotonic (DM), and Earliest Deadline First (EDF) scheduling algorithms. By integrating theoretical methods such as response-time analysis for RM and DM, and both utilization-based and demand-based analyses for EDF, it evaluates task feasibility under implicit and constrained deadline scenarios. The modular Python implementation ensures extensibility and precise analysis, facilitating the exploration of schedulability across varying task parameters and operational constraints.

CCS CONCEPTS • Schedulability analysis • Real-time system

Additional Keywords and Phrases: Utilization bound, Response time analysis, Processor Demand Criterion

1 INTRODUCTION

Real-time systems operate under stringent timing constraints, requiring schedulability analysis to guarantee task completion within defined deadlines. These systems rely on well-defined scheduling policies, each with unique methods for prioritizing tasks and managing deadlines. This report examines four fundamental schedulability tests: response-time analysis for Rate Monotonic (RM) and Deadline Monotonic (DM) scheduling, and utilization-based and demand-based analyses for Earliest Deadline First (EDF) scheduling.

Building upon and advancing beyond the simulation framework from Homework 2, this report adopts a more sophisticated and systematic analytical approach. Unlike simulation-based evaluations over a fixed time window, the schedulability analysis here directly validates task sets without relying on extensive runtime simulations, ensuring precise and efficient feasibility determination. Leveraging Python's flexibility and precision, the report provides a comprehensive evaluation of task sets under various constraints, highlighting the strengths and limitations of different scheduling techniques.

1.1 Architecture Overview

The system follows a modular architecture centered around the `TasksetAnalyzer` class. This design choice enables:

- Easy extension for new scheduling algorithms
- Separation of concerns between analysis methods

- Efficient task set management for multiprocessing

2 CORE IMPLEMENTATION DETAILS

2.1 Priority Assignment by sorting

```
if scheduling_algorithm == "RM":
    self.task_set = sorted(task_set, key=lambda task: (task.period, task.index))
elif scheduling_algorithm == "DM":
    self.task_set = sorted(
        task_set, key=lambda task: (task.relative_deadline, task.index)
    )
else:
    self.task_set = task_set
```

The initialization phase implements priority assignment through sorting:

- For RM: Tasks are sorted by period (rate-monotonic principle)
- For DM: Sorting is based on relative deadlines
- For EDF: No static sorting is needed due to Processors Demand Criterion doesn't use priority for analysis.

The secondary sort key (`task.index`) ensures deterministic behavior for tasks with equal periods or deadlines, preventing priority inversions. Also, priority is needed only for RM and DM because Response Time Analysis Method only use priority information for analysis.

2.2 Schedulability Analysis

Each schedulability analysis has been implemented according to the corresponding algorithm.

2.2.1Response Time Analysis

$$R_i = C_i + \sum_{\text{all higher Priority tasks } \tau_j} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

$If\ R_i \leq D_i$ for all tasks, task set is schedulable

```
for i, task in enumerate(self.task_set):
    R_previous = task.wcet
```

```
        while True:
            interference = sum(
                math.ceil(R_previous / HP_task.period) * HP_task.wcet
                for HP_task in self.task_set[:i])
            R_current = task.wcet + interference
            if R_current > task.relative_deadline:
                return "F"
            elif R_current == R_previous:
                break
            R_previous = R_current
    return "P"
```

2.2.2EDF Utilization Bound Analysis

$$If\ Sum\ of\ Utilization \leq 1.0, task\ set\ is\ schedulable$$

```
    return "P" if sum(task.utilization for task in self.task_set) <= 1 else "F"
```

2.2.3EDF Processor Demand Criterion

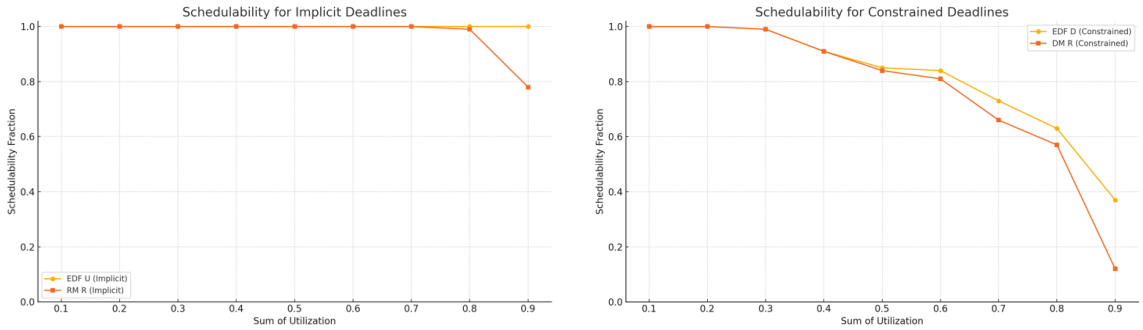
$$g(0,L) = \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i$$

If g(0,L) $\leq$ *L for all interval L > 0, task set is schedulable*

```
    interval_candidates = sorted( {
            task.relative_deadline + k * task.period
            for task in self.task_set
            for k in range(hyperperiod // task.period + 1)
            # min(hyperperiod, l_start)
            if task.relative_deadline + k * task.period <= upper_bound
        })
    for interval in interval_candidates:
        total_demand = sum(
            math.floor(
                (interval - task.relative_deadline + task.period) / task.period) * task.wcet
            for task in self.task_set
```

```
        )
        if total_demand > interval:
            return "F"
    return "P"
```

3 RESULT



The input files include task sets with three tasks each, where the sum of utilization ranges from 0.1 to 0.9 in steps of 0.1. Both implicit and constrained deadline scenarios are tested. The results, shown in Figures 1 and 2, were derived using Python-based implementations of utilization- and response-time analyses.

3.1 Implicit Deadlines

Figure 1 demonstrates the schedulability of task sets under implicit deadlines for Earliest Deadline First (EDF) and Rate Monotonic (RM) scheduling algorithms. For EDF, the utilization-based analysis (EDF U) achieves perfect schedulability up to the theoretical bound of 1.0, confirming EDF's efficiency in implicit deadline scenarios. In contrast, RM, using response-time analysis (RM R), maintains schedulability up to approximately 0.9 utilization. This behavior aligns with RM's utilization bound for three tasks, calculated as $U_{lub}^{RM} = 3\left(2^{\frac{1}{3}} - 1\right) \approx 0.779$, where n = 3. Interestingly, when the sum of utilization is 0.8, it exceeds the theoretical RM utilization bound. However, response-time analysis reveals that task sets remain schedulable despite this. This demonstrates that response-time analysis is more precise than the utilization-bound test, as it accounts for task interactions and their specific timing constraints rather than relying on generalized bounds.

3.2 Constrained Deadlines

Figure 2 highlights the results for constrained deadlines, analyzed using EDF demand-based analysis (EDF D) and Deadline Monotonic response-time analysis (DM R). EDF outperforms DM at most utilization levels, retaining a higher schedulability fraction. However, both DM and EDF experience a sharp decline in schedulability fraction starting at a utilization of 0.8. Beyond this point, EDF continues to demonstrate more robust performance compared to DM, though its efficiency decreases significantly as utilization approaches 1.0 due to increased computational demand. DM shows lower overall schedulability but maintains stable performance under specific conditions due to the simplicity of its fixed-priority approach. ($\leq$ 0.8)