# Rate Monotonic vs. EDF: Judgment Day

**von Buttazzo, G. C.**

Dominik Schlecht

May 4, 2015

## Contents

# 1 Einleitung

## 1.1 Meta-Informationen

- Author:
  - Giorgio C. Buttazzo
  - University of Pavia, Italien
  - buttazzo@unipv.it
- Whitepaper
  - Rate Monotonic vs. EDF: Judgment Day
  - Real-Time Systems, 29, 5–26, 2005

## 1.2 Abstract

Since the first results published in 1973 by Liu and Layland on the Rate Monotonic (RM) and Earliest Deadline First (EDF) algorithms, a lot of progress has been made in the schedulability analysis of periodic task sets. Unfortunately, many misconceptions still exist about the properties of these two scheduling methods, which usually tend to favor RM more than EDF . Typical wrong statements often heard in technical conferences and even in research papers claim that RM is easier to analyze than EDF, it introduces less runtime overhead, it is more predictable in overload conditions, and causes less jitter in task execution.Since the above statements are either wrong, or not precise , it is time to clarify these issues in a systematic fashion, because the use of EDF allows a better exploitation of the available resources and significantly improves system's performance. This paper compares RM against EDF under several aspects, using existing theoretical results, specific simulation experiments, or simple counterexamples to show that many common beliefs are either false or only restricted to specific situations.[1]

## 1.3 Flashback - Scheduling

Wir definieren:

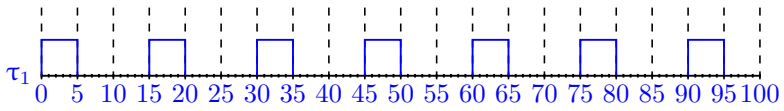
$$\tau_{i,k} \text{ als Job mit einer absoluten Deadline } d_{i,k} \tag{1}$$

$$\tau_i \text{ als infinite Folge von Jobs } \tau_{i,k} \text{ mit} \tag{2}$$

$$\begin{array}{ll} \text{Wort-Case-Execution-Time} & C_i \\ \text{Task-Period} & T_i \end{array} \tag{3}$$

Beispiel mit einem Task:

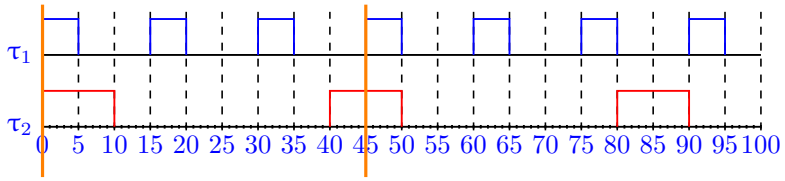
| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 5 | 15 |




[1] Rate Monotonics vs. EDF: Judgment Day, Girorgio C. Buttazzo, 2005

Beispiel mit 2 Tasks:

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 5 | 15 |
| $\tau_2$ | 10 | 40 |




Problem: 2 Tasks versuchen gleichzeitig auf einem Prozessor zu laufen.

# 2 Rate Monotonic & Earliest Deadline First

## 2.1 Grundlagen

Rate Monotonics:

- Task $T_i$ mit kürzester Periode wird bevorzugt.
- Task $T_i$ wird anfangs eine Priorität zugewiesen.

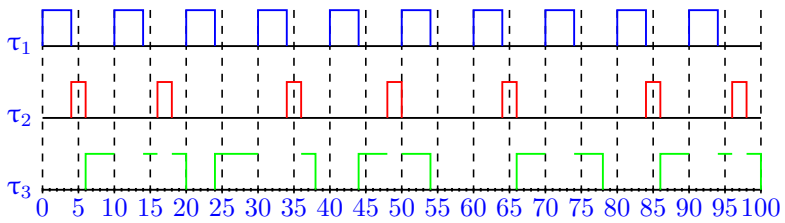
Earliest Deadline First:

- Job $T_{i,k}$ mit der nächsten Deadline wird bevorzugt.
- In diesem Paper ist die Deadline gleich mit der Periode.
- Die Priorität von Task $T_i$ entscheidet sich während der Laufzeit.

## 2.2 Beispiele

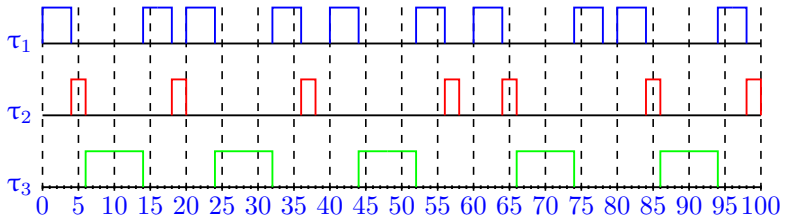
### Beispiel Rate Monotonics

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 4 | 10 |
| $\tau_2$ | 2 | 16 |
| $\tau_3$ | 8 | 20 |



**Beispiel Earliest Deadline First**

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 4 | 10 |
| $\tau_2$ | 2 | 16 |
| $\tau_3$ | 8 | 20 |




# 3 Vergleich

## 3.1 Implementation Complexity

**Mythos:**

- Rate Monotonics ist einfacher zu implementieren als Earliest Deadline First.

**Fakt:**

- Auf einem kommerziellen Kernel mit festen Prioritätsleveln ist Rate Monotonics einfacher zu implementieren.

**Weitere Faktoren:**

- Wird auf einem bestehenden System entwickelt?
- Sind die Prioritäten festgesetzt oder können diese während der Laufzeit verändert werden?
- Wie viele Prioritäts-Level gibt es?

**Annahme:**

- Das System wird von Grund auf mit einer Ready-Queue implementiert.
- In dieser werden die Tasks für Rate Monotonics
  - absteigend nach nach dem Prioritäten-Level

  und für Earliest Deadline First
  - aufsteigend nach der absoluten Deadline

  gespeichert.

**Fazit:**

- Unter den richtigen Vorbedingungen ist auch EDF leicht zu implementieren.

## 3.2 Runtime Overhead

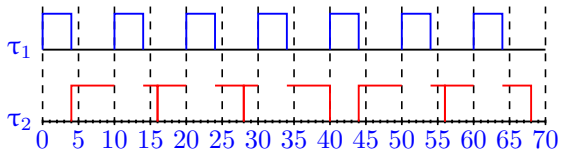
**Mythos:**

- Rate Monotonics produziert weniger Runtime-Overhead, da die Prioritäten währen der Laufzeit nicht neu berechnet werden müssen.

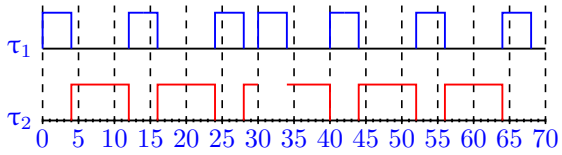
**Beispiel**

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 4 | 10 |
| $\tau_2$ | 8 | 14 |

- Rate Monotonics:




- Earliest Deadline First:

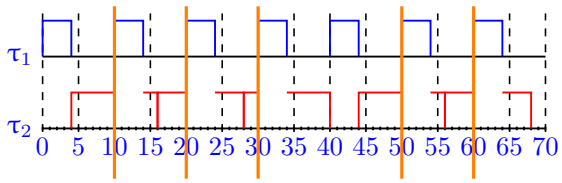



**Context-Switching/Preemptions:**

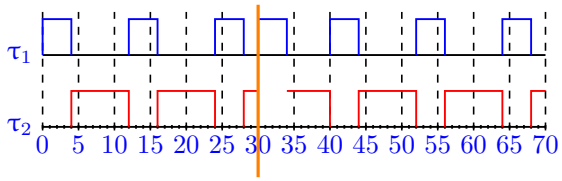
- Umschalten zwischen verschiedenen Tasks.
- Zieht Aufwände mit sich.

**Vergleich Rate Monotonics vs. Earliest Deadline First**

- Rate Monotonics:




- Earliest Deadline First:




**Fazit:**

- Beachtet man den Aufwand der Context-Switches, erzeugt Rate Monotonics mehr Overhead als Earliest Deadline First

## 3.3 Schedulability Analysis

**Erklärung:**
Schedulability meint, dass eine Menge von periodischen Task mithilfe eines Algorithmus planbar ist.

**Mythos:**

- Die Einteilung ist bei Rate Monotonics leichter berechenbar als bei Earliest Deadline First.

**Allgemein gilt:**

$$U_i = C_i/T_i \tag{4}$$

Desweiteren gilt, dass ein Task-Set $P$ unter RM nur sicher planbar seien kann, wenn

$$\prod_{i=0}^{n}(U_i + 1) \leq 2 \tag{5}$$

und unter EDF nur (und auch wirklich nur) planbar sein, wenn

$$\sum_{i=1}^{n} U_i \leq 1 \tag{6}$$

**Beispiel:**

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 1 | 4 |
| $\tau_2$ | 3 | 8 |
| $\tau_3$ | 2 | 16 |

- RM: $U = (\frac{1}{4}+1)(\frac{3}{8}+1)(\frac{2}{16}+1) \approx 1.93 \leq 2$
- EDF: $U = \frac{1}{4}+\frac{3}{8}+\frac{2}{16} = \frac{3}{4} = 0.75 \leq 1$

### RTA und PDC

Response Time Analysis (RTA) Algorithmus für Rate Monotonics:

$$D_i \geq \begin{cases} R_i^{(0)} = C_i \\ R_i^{(k)} = C_i + \sum_{j:D_j<D_i} \lceil \frac{R_i^{k-1}}{T_j} \rceil C_j \end{cases} \tag{7}$$

Processor Demand Criterion (PDC) Algorithmus für Earliest Deadline First:

$$\forall L > 0,\ \sum_{i=1}^{n} \lfloor \frac{L + T_i - D_i}{T_i} \rfloor C_i \leq L \tag{8}$$

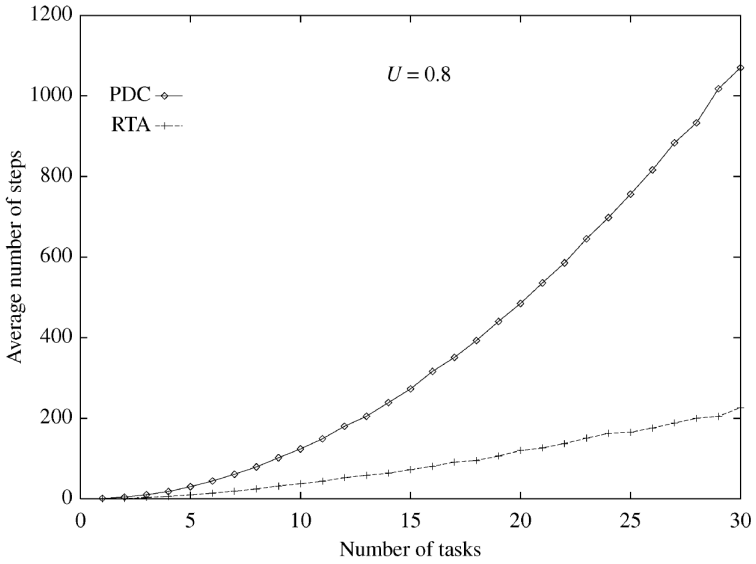



Figure 1: Vergleich RTA vs. PDC [2]

**Fazit:**

- Komplexität für Rate Monotonics: pseudo-polynomial
- Komplexität für Earliest Deadline First:
  - pseudo-polynomial
  - in besonderen Fällen $O(n)$
- Bei einer hohen Anzahl von Tasks ist Rate Monotonics besser zu berechnen (siehe Figure 1 auf Seite 7).
- Bei Earliest Deadline First ist für höhere Auslastungen ein garantiertes Scheduling möglich.

## 3.4 Robustness During Overloads

**Mythos:**

- Rate Monotonics ist in Overload-Situationen besser vorhersehbar.

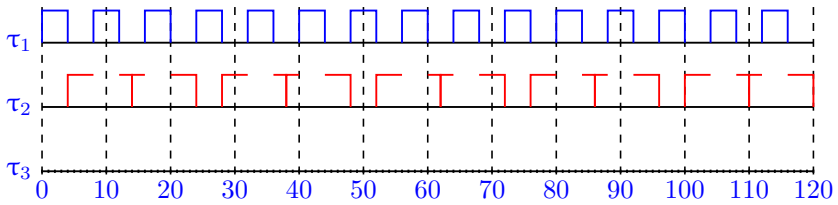
### 3.4.1 Permanent Overload

**Permanent Overload: Taskset**

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 4 | 8 |
| $\tau_2$ | 6 | 12 |
| $\tau_3$ | 5 | 20 |

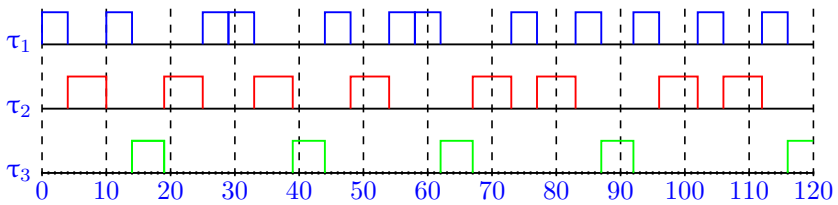
**Permanent Overload: Rate Monotonics**

- $U = (\frac{4}{8}+1)(\frac{6}{12}+1)(\frac{5}{20}+1) \approx 2.81 \not\leq 2$




**Permanent Overload: Earliest Deadline First**

- $U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25 \not\leq 1$



**Vergleich Permanent Overload**

Rate Monotonics:

- Tasks mit langer Periode werden vollständig blockiert!
- Gut vorhersagbar.

Earliest Deadline First:

- Sieht chaotischer aus.
- Durchschnittliche Periode $\bar{T}_i$ für einen Task $\tau_i$ ist gegeben durch

$$\bar{T}_i = T_i \cdot U \tag{9}$$

**Fazit Permanent Overload:**

- Beide Verfahren bei permanenter Überlastung gut vorhersagbar.
- Einsatzgebiet ist stark Situationsabhängig.

### 3.4.2 Transient Overload

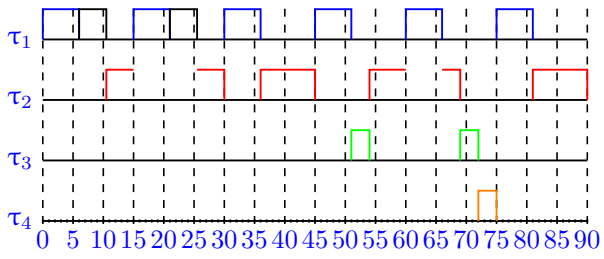
**Annahme für Rate Monotonics:**

- Es werden Tasks mit kurzen Perioden bevorzugt.

⇒ Falls ein Task seine Deadline überschreitet, wird der Task mit der längsten Periodenlänge verschoben/unterbrochen .

**Gegenbeispiel**

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 6 | 15 |
| $\tau_2$ | 9 | 27 |
| $\tau_3$ | 3 | 60 |
| $\tau_4$ | 3 | 90 |




⇒ $\tau_2$ wird unterbrochen, während Task $\tau_3$ und $\tau_4$ nicht beeinflusst werden!

**Fazit:**

- Permanent Overload: Gleichwertig.
- Transient Overload:
  - Rate Monotonics verführt zu falschen Annahmen.

## 3.5 Jitter and Latency

### Absolute Response Time Jitter

Absolute Response Time Jitter $ARJ_i$ ist definiert durch

$$ARJ_i = maxR_{i,k} - minR_{i,k} \tag{10}$$

mit $R_{i,K}$ als Response-Time für den $k$-ten Job von $\tau_i$.

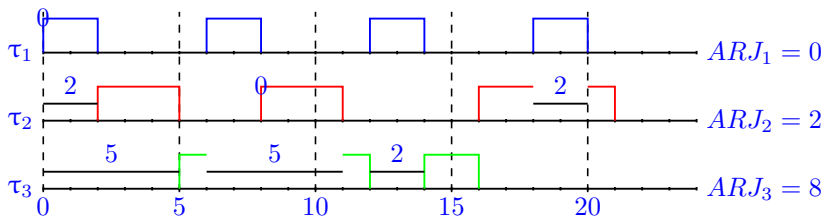
### Mythos:

- Durch die festen Prioritäten entsteht währen der Laufzeit bei Rate Monotonics weniger Jitter als bei Earliest Deadline First.

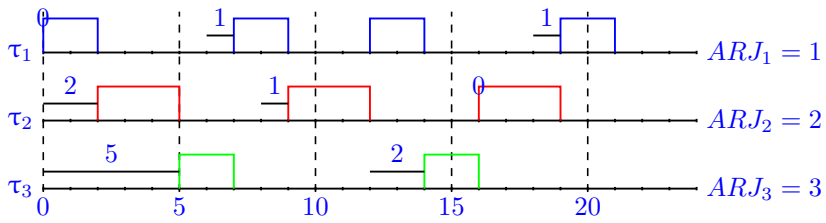
### Beispiel Taskset

| Task ($\tau_i$) | Dauer ($C_i$) | Task-Periode ($T_i$) |
|---|---|---|
| $\tau_1$ | 2 | 6 |
| $\tau_2$ | 3 | 8 |
| $\tau_3$ | 2 | 12 |

### Beispiel Jitter Rate Monotonics




### Beispiel Jitter Earliest Deadline First




### Fazit:

- RM hält Jitter für die hoch priorisierten Tasks sehr niedrig, vernachlässigt jedoch die anderen Tasks.
- Insgesamt erzeugt EDF, gerade bei hoher Auslastung, wesentlich weniger Jitter (siehe Figure 2 auf Seite 11).

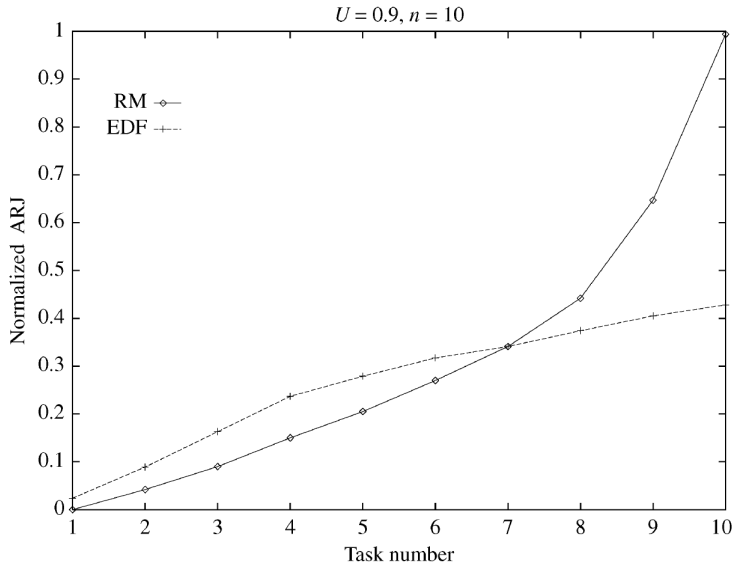



Figure 2: Vergleich ARJ bei RM vs EDF[3]

# 4 Fazit

**Vorteile von Rate Monotonics:**

- leichtere Implementierung in kommerziellen Systemen
- RTA benötigt weniger Schritte als PDC

**Vorteile Earliest Deadline First:**

- Erlaubt vertrauenswürdiges Scheduling solange $U \leq 1$
- Weniger Runtime Overhead

**Unentschieden oder Situationsabhängig:**

- Overload Situations
- Jitter Control

Dies Zeigt, dass Rate Monotonics Earliest Deadline First nicht überlegen ist.