Rate Monotonic vs. EDF: Judgment Day

von Buttazzo, G. C.

Dominik Schlecht

Technische Hochschule Ingolstadt

April 27, 2015

Gliederung

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

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

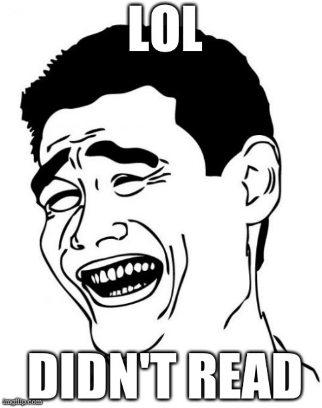
Gliederung

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

Abstract



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.[2]

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

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.[3]

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

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.[4]

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

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.[5]

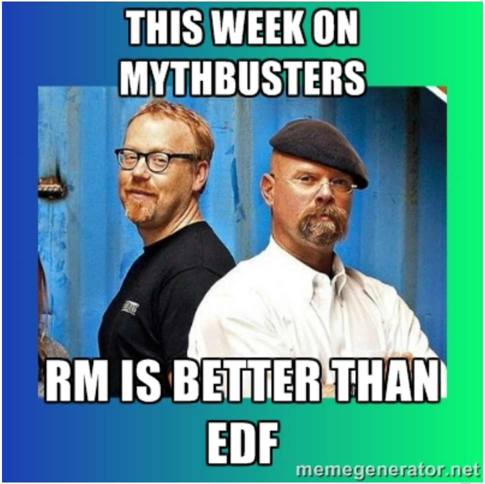
[5] Rate Monotonics vs. EDF: Judgment Day, Girorgio C. Buttazzo, 2005

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.[6]

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

Abstract


THIS WEEK ON
MYTHBUSTERS
RM IS BETTER THAN
EDF
memegenerator.net

Gliederung

Flashback - Scheduling

Wir definieren:

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

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

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

Flashback - Scheduling

Wir definieren:

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

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

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

Flashback - Scheduling

Wir definieren:

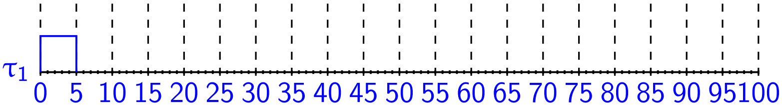
$$\tau_{i,k} \text{ als Job mit einer absoluten Deadline } d_{i,k} \tag{1}$$

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

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

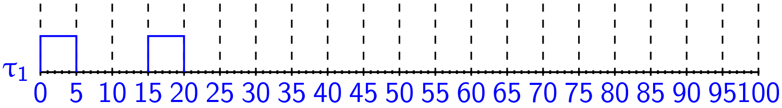
Flashback - Scheduling

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15



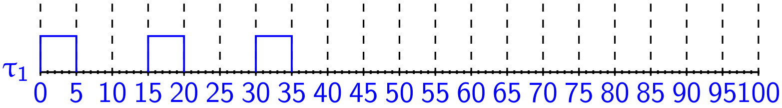
Flashback - Scheduling

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15



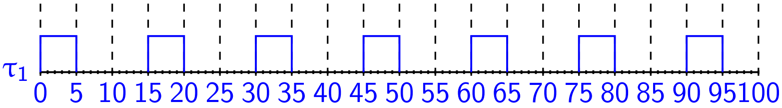
Flashback - Scheduling

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15



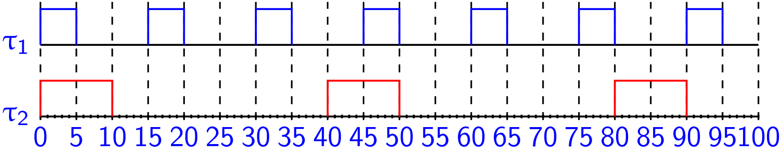
Flashback - Scheduling

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15



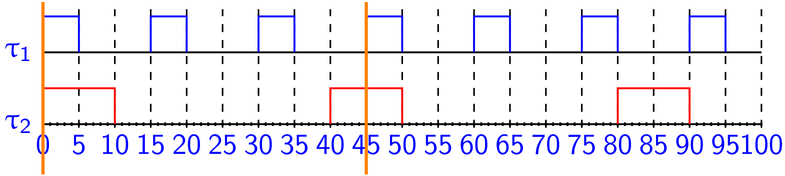
Flashback - Scheduling

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	10	40



Flashback - Scheduling

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	10	40



Gliederung

Grundlagen

Rate Monotonics:

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

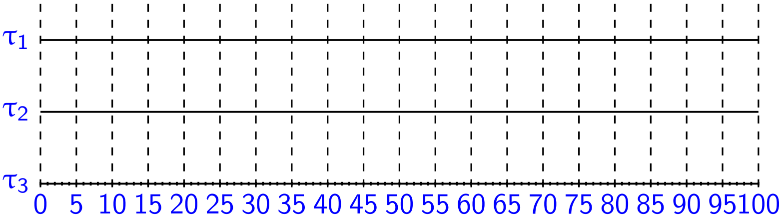
Erliest Deadline First:

- Job $T_{i,k}$ mit der nächsten Deadline wird bevorzugt
- Die Priorität von Task T_i entscheidet sich während der Laufzeit

Gliederung

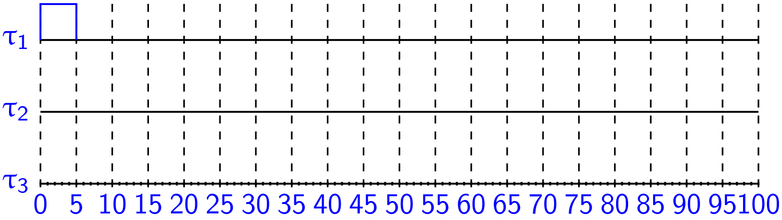
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



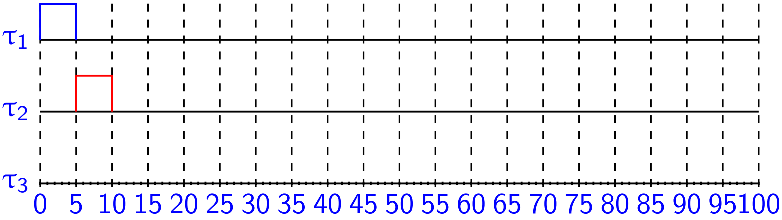
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



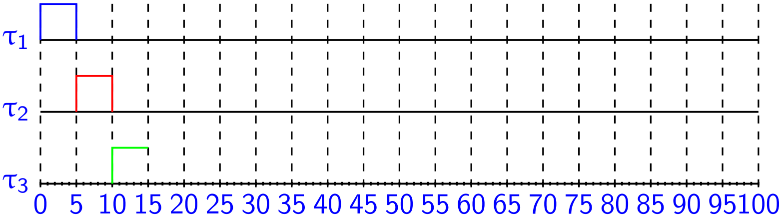
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



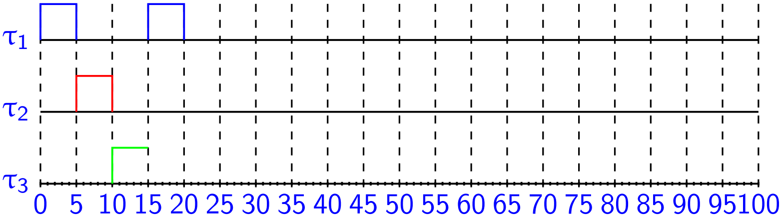
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



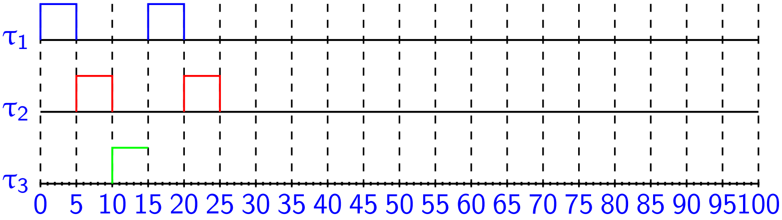
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



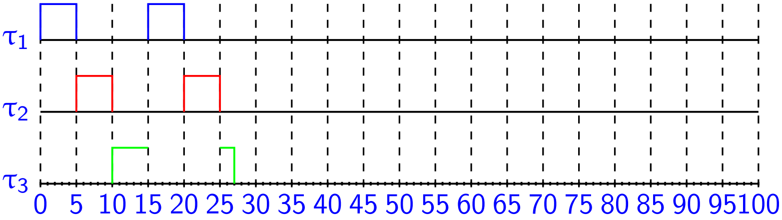
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



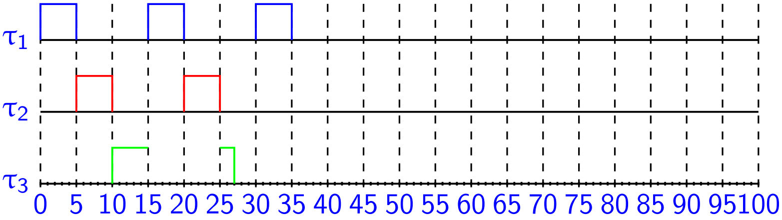
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



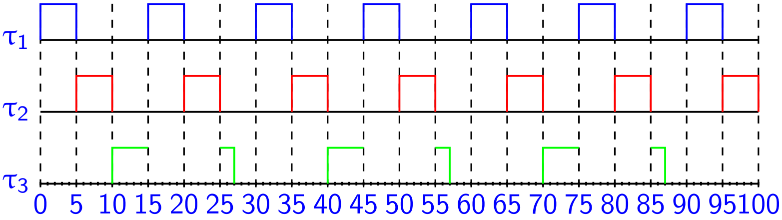
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



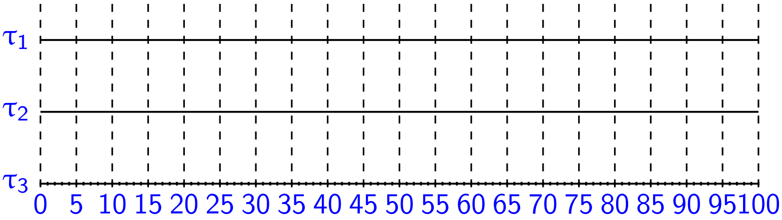
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



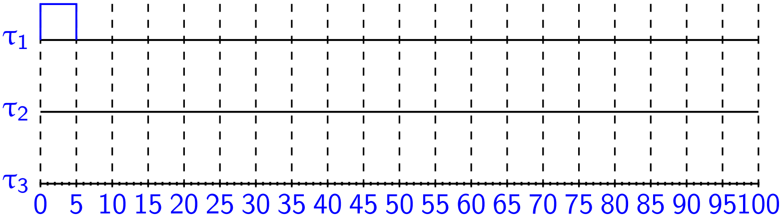
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



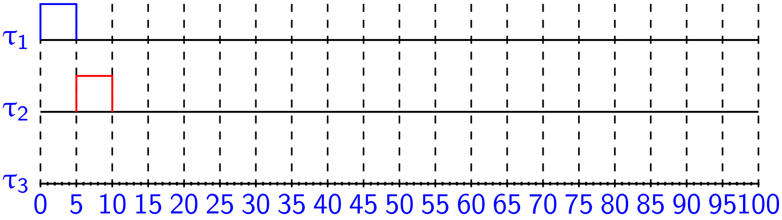
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



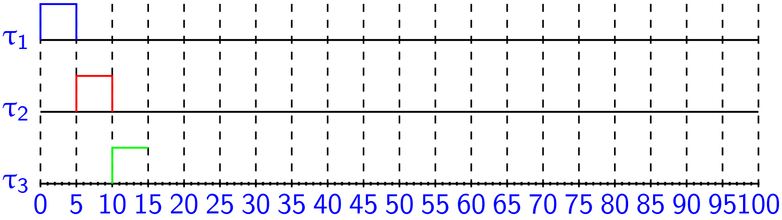
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



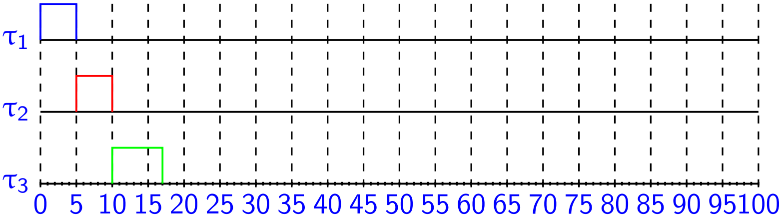
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



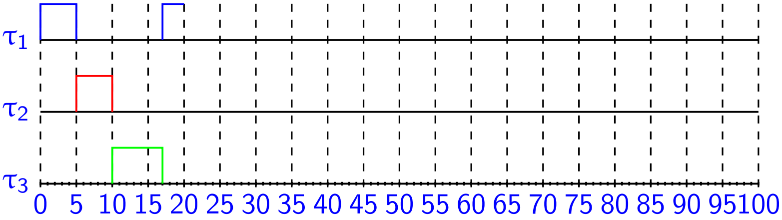
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



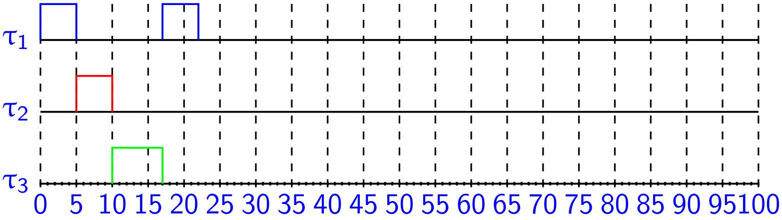
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



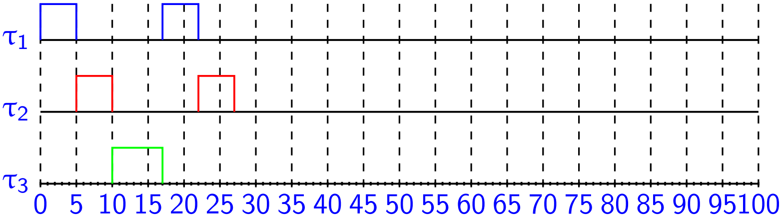
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



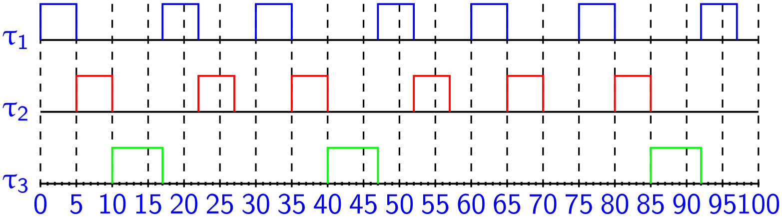
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	5	15
τ_2	5	20
τ_3	7	40



Vergleich

Gliederung

Implementation Complexity

Mythos:

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

Implementation Complexity

Fakt:

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

Ist es so einfach?

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?

Implementation Complexity

Fakt:

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

Ist es so einfach?

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?

Implementation Complexity

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 Erliest Deadline First
  - aufsteigend nach der absoluten Deadline

  gespeichert.

Implementation Complexity

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 Erliest Deadline First
  - aufsteigend nach der absoluten Deadline

  gespeichert.

Implementation Complexity

Fazit:

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

Implementation Complexity

Gliederung

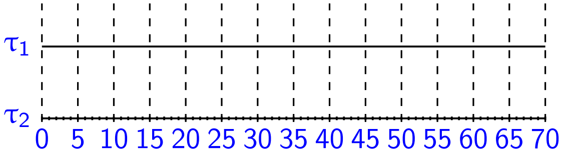
Runtime Overhead

Mythos:

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

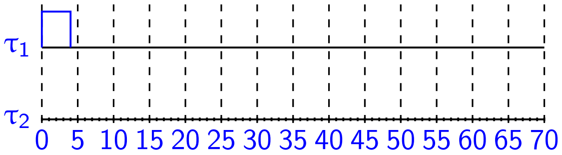
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



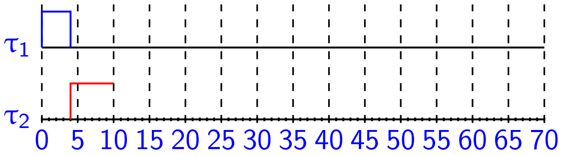
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



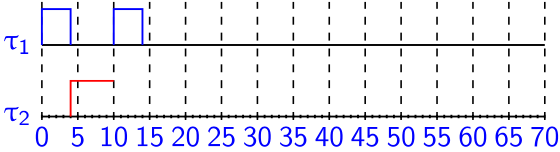
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



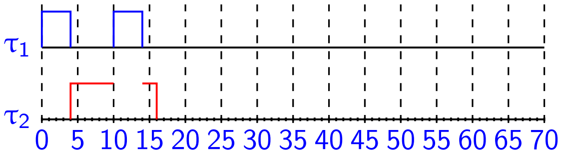
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



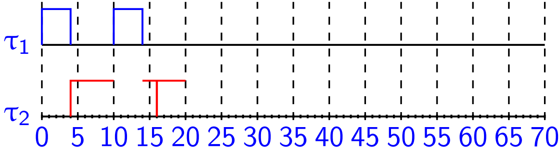
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



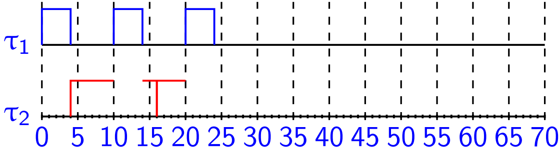
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



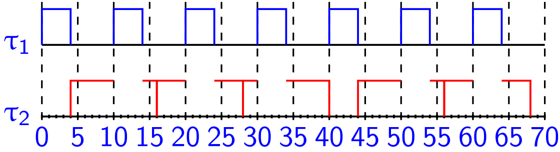
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



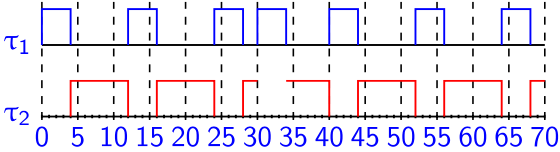
Beispiel Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14



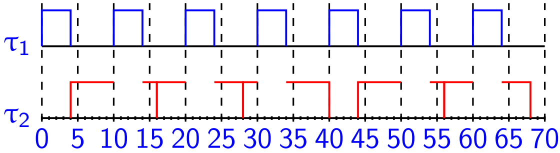
Beispiel Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	10
τ_2	8	14

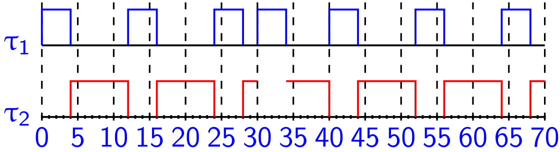


Runtime Overhead

Rate Monotonics:



Erliest Deadline First:



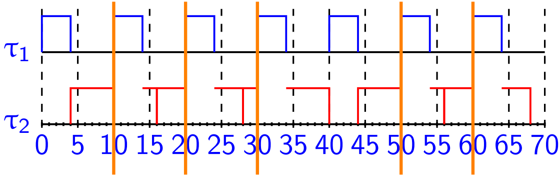
Runtime Overhead

Context-Switching/Preemptions:

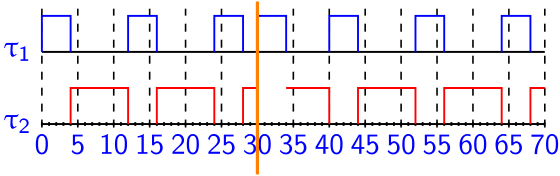
- Umschalten zwischen verschiedenen Tasks.
- Zieht Aufwände mit sich.

Runtime Overhead

Rate Monotonics:



Erliest Deadline First:



Runtime Overhead

Fazit:

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

Runtime Overhead

Gliederung

Schedulability Analysis

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

Mythos:

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

Schedulability Analysis

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

Mythos:

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

Schedulability Analysis

Allgemein:

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

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

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

Schedulability Analysis

Beispiel:

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	1	4
τ_2	3	8
τ_3	2	16

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

EDF: $U = \frac{1}{4}+\frac{3}{8}+\frac{2}{16} = \frac{3}{4} = 0.75 \leq 1$

Schedulability Analysis

Beispiel:

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	1	4
τ_2	3	8
τ_3	2	16

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

EDF: $U = \frac{1}{4} + \frac{3}{8} + \frac{2}{16} = \frac{3}{4} = 0.75 \leq 1$

Schedulability Analysis

Beispiel:

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	1	4
τ_2	3	8
τ_3	2	16

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

EDF: $U = \frac{1}{4} + \frac{3}{8} + \frac{2}{16} = \frac{3}{4} = 0.75 \leq 1$

Schedulability Analysis

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 Erliest Deadline First:

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


I HAVE NO
IDEA WHAT
I'M DOING

Schedulability Analysis

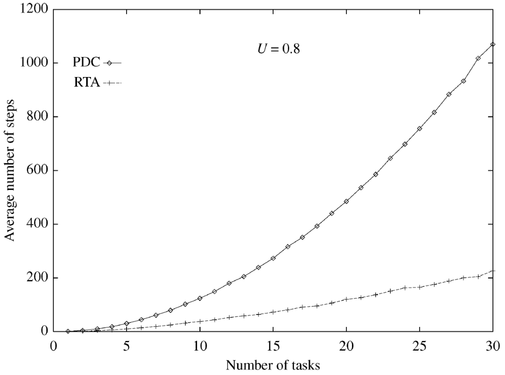


Figure: Vergleich RTA vs. PDC [7]

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

Schedulability Analysis

Fazit:

- Komplexität für Rate Monotonics: pseudo-polynomial
- Komplexität für Erliest Deadline First:
  - pseudo-polynomial
  - in besonderen Fällen $O(n)$
- Bei einer hohen Anzahl von Tasks ist Rate Monotonics besser zu berechnen (mit Ausnahmen)
- Bei Erliest Deadline First ist für höhere Auslastungen ein garantiertes Scheduling möglich

Schedulability Analysis

Schedulability Analysis

Gliederung

Robustness During Overloads

Mythos:

- Rate Monotonics ist in Overload-Situationen besser vorhersehbar.

Szenarien:

- Permanent Overload
- Transient Overload

Robustness During Overloads

Mythos:

- Rate Monotonics ist in Overload-Situationen besser vorhersehbar.

Szenarien:

- Permanent Overload
- Transient Overload

Permanent Overload: Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	8
τ_2	6	12
τ_3	5	20

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

Permanent Overload: Rate Monotonics

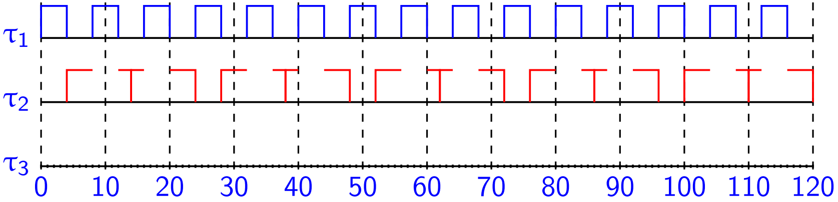
Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	8
τ_2	6	12
τ_3	5	20

$$U = (\frac{4}{8} + 1)(\frac{6}{12} + 1)(\frac{5}{20} + 1) \approx 2.81 \nleq 2$$

Permanent Overload: Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	8
τ_2	6	12
τ_3	5	20

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



Permanent Overload: Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	8
τ_2	6	12
τ_3	5	20

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$

Permanent Overload: Erliest Deadline First

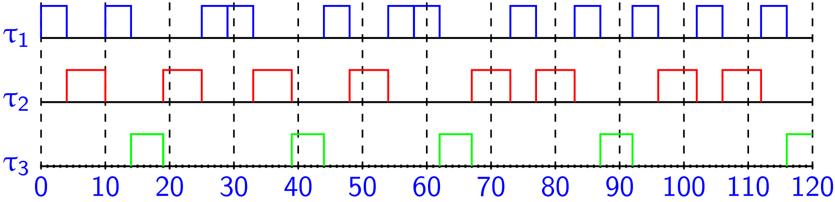
Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	8
τ_2	6	12
τ_3	5	20

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$

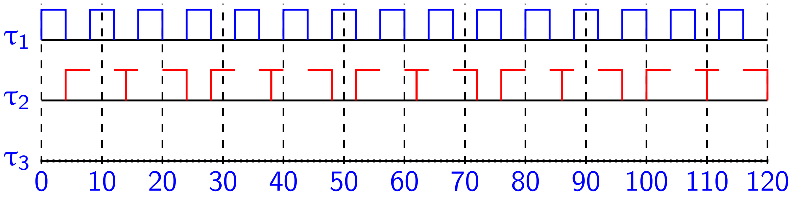
Permanent Overload: Erliest Deadline First

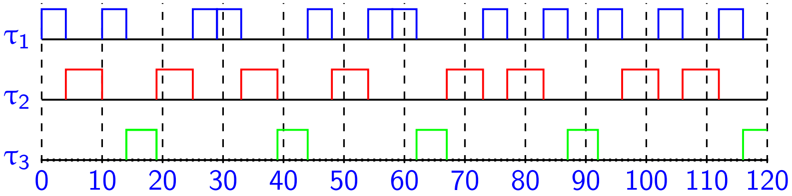
Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	4	8
τ_2	6	12
τ_3	5	20

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



Permanent Overload





Permanent Overload

Rate Monotonics:

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

Erliest Deadline First:

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

$$\bar{T}_i = T_i \cdot U \qquad (9)$$

Permanent Overload

Fazit:

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

Transient Overload

Annahme für RM:

- Es werden Tasks mit kurzen Perioden bevorzugt.

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

Transient Overload

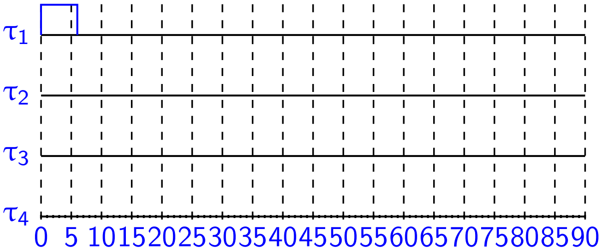
Annahme für RM:

- Es werden Tasks mit kurzen Perioden bevorzugt.

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

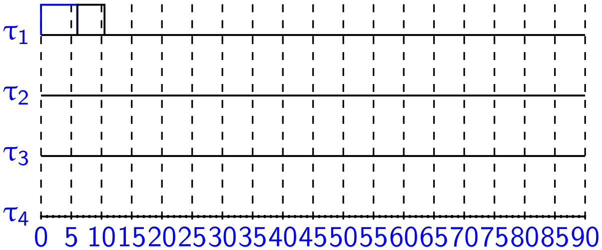
Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



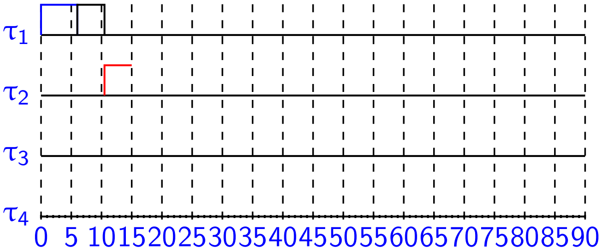
Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



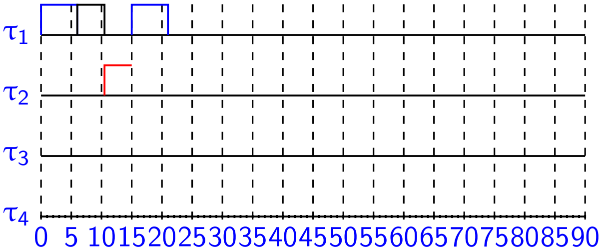
Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



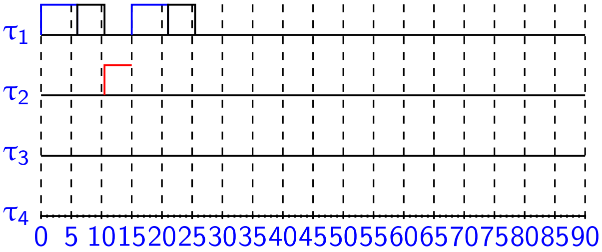
Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



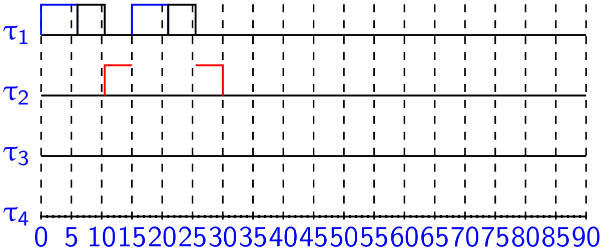
Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



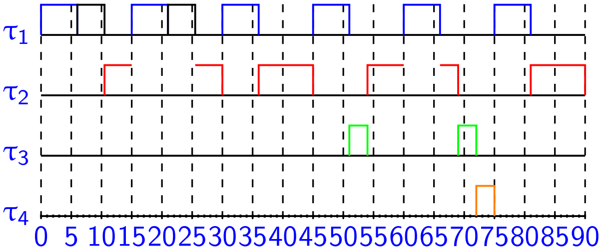
Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



Transient Overload

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	6	15
τ_2	9	27
τ_3	3	60
τ_4	3	90



Robustness During Overloads

Fazit:

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

Robustness During Overloads

Gliederung

Jitter and Latency

Definition von 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 τ_i.

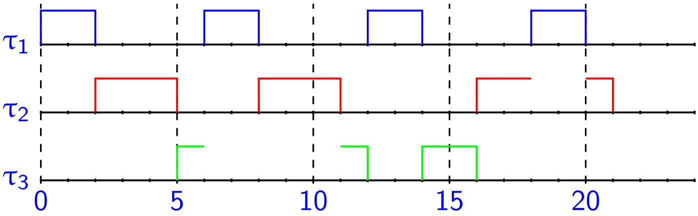
Jitter and Latency

Mythos:

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

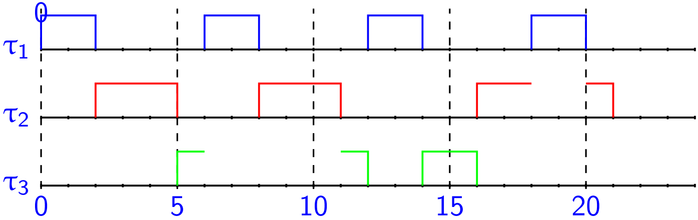
Beispiel Jitter Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



Beispiel Jitter Rate Monotonics

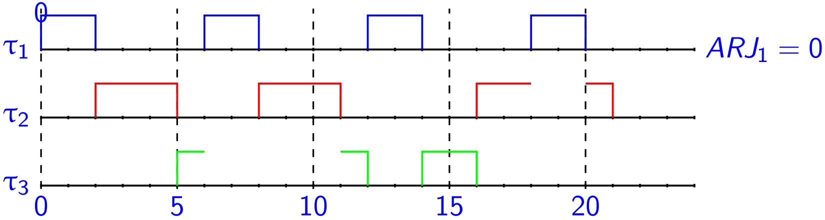
Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



$ARJ_1 =$

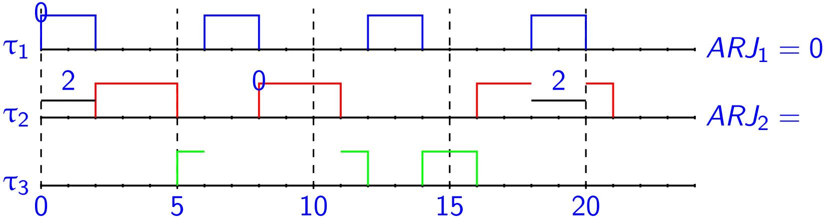
Beispiel Jitter Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



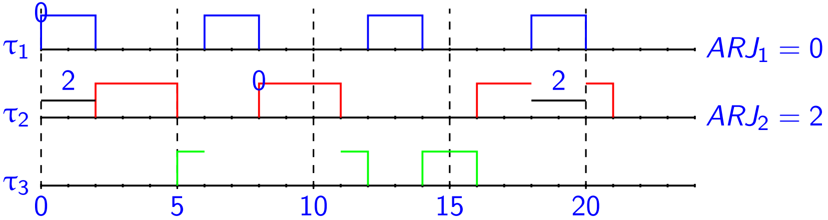
Beispiel Jitter Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



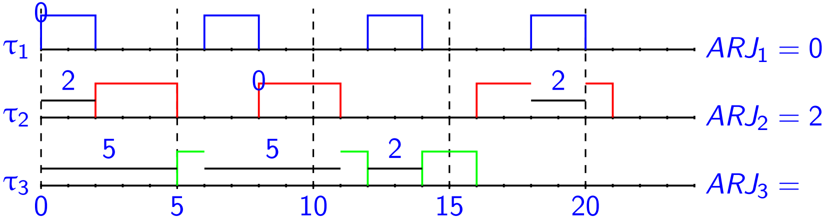
Beispiel Jitter Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



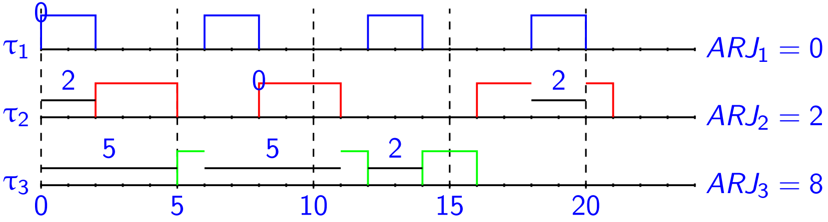
Beispiel Jitter Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



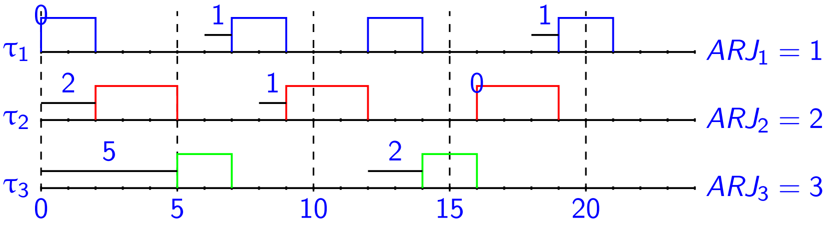
Beispiel Jitter Rate Monotonics

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



Beispiel Jitter Erliest Deadline First

Task (τ_i)	Dauer (C_i)	Task-Periode (T_i)
τ_1	2	6
τ_2	3	8
τ_3	2	12



Jitter and Latency

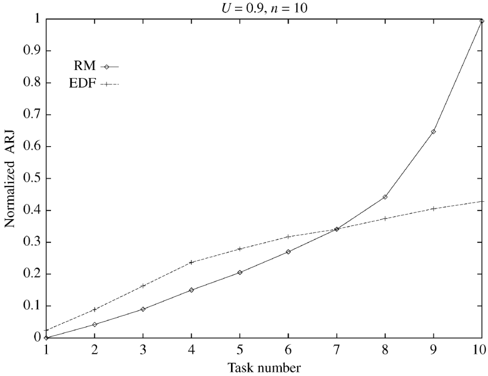


Figure: Vergleich ARJ bei RM vs EDF[8]

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

Jitter and Latency

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.

Jitter and Latency

Gliederung

Fazit

Vorteile von Rate Monotonics:

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

Vorteile Erliest Deadline First:

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

Unentschieden oder Situationsabhängig:

- Overload Situations
- Jitter Control

Fazit

Vorteile von Rate Monotonics:

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

Vorteile Erliest Deadline First:

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

Unentschieden oder Situationsabhängig:

- Overload Situations
- Jitter Control

Fazit

Vorteile von Rate Monotonics:

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

Vorteile Erliest Deadline First:

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

Unentschieden oder Situationsabhängig:

- Overload Situations
- Jitter Control

Fazit

Vorteile von Rate Monotonics:

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

Vorteile Erliest Deadline First:

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

Unentschieden oder Situationsabhängig:

- Overload Situations
- Jitter Control

Fazit

Vorteile von Rate Monotonics:

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

Vorteile Erliest Deadline First:

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

Unentschieden oder Situationsabhängig:

- Overload Situations
- Jitter Control

Fazit

Vorteile von Rate Monotonics:

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

Vorteile Erliest Deadline First:

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

Unentschieden oder Situationsabhängig:

- Overload Situations
- Jitter Control

Fazit

Fragen?