Online scheduling of aperiodic tasks

Buttazzo’s book, chapter 3

Paper:

Disclaimer: A few slides have been taken from Giorgio Buttazzo’s website:
http://retis.sssup.it/~giorgio/rts-MECS.html

Thank you Giorgio!
Agenda

1. **Online scheduling algorithms for real-time systems** (chapter 3):
   - What do we cover?
     - **EDF**
       - An interesting yet long “proof of optimality” (from 1974)

2. **Explanation of the Lab assignments**
3. **Table-driven scheduling**
4. **Non-preemptive scheduling**
   - **Proof of non-optimality of EDF** (from 2016)
Recall: absolute deadline v.s. relative deadline

Absolute deadline

Relative deadline

Response time: \[ R_i \]

Absolute deadline

Maximum lateness \[ L_{\text{max}} = \max \{ L_i \mid \forall \tau_i \} \]

If \[ L_{\text{max}} \leq 0 \], then there is no deadline miss
Earliest deadline first (**EDF**) 

**Algorithm** [Horn 74]
- Order the ready queue by increasing absolute deadline.

**Assumptions**
- Horn’s algorithm is preemptive and is for independent tasks
  Note: the “**EDF**” that is known today (2019) can be preemptive or non-preemptive.
- dynamic ($d_i$ depends on the activation time of the task)

**Property**
- Under the assumptions above, EDF minimizes

  the maximum lateness ($L_{max}$)
EDF exercise (3 min)

Check whether EDF produces a feasible schedule

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$\alpha_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>4</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>5</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

What is the maximum lateness?
EDF exercise (3 min)

Task 1

Task 2

Task 3

Task 4

Note the difference between $D_i$ and $d_i$
How can we design an online test for EDF that tells if we can safely add a new task to the system at time $t$?

This is called an online admission test.

**Safely** = not causing a deadline miss for the tasks that execute in the system?
An online admission test for EDF

How should we make a test?

At time $t = 7$ should we let Task 4 be added to the system?

$c_i(t)$ denotes the remained execution time of Task $i$ at time $t$
An online admission test for EDF

1- Sort and **re-index** tasks by their deadlines (only tasks with \( d_i \geq t \))

2- Check if there is enough time to finish the tasks for each of the deadlines

At time \( t = 7 \) should we let Task 4 be added to the system?

\[
\forall i \sum_{k=1}^{i} c_k(t) \leq d_i - t
\]

**Important note**: the equation above works only if you have re-labeled your tasks after sorting them by their absolute deadline!
An online admission test for EDF

1- Sort and re-index tasks by their deadlines (only tasks with $d_i \geq t$)
2- Check if there is enough time to finish the tasks for each of the deadlines

At time $t = 7$ should we let Task 4 be added to the system?

\[ \forall i \sum_{k=1}^{i} c_k(t) \leq d_i - t \]

- $i = 1 \Rightarrow c_1(t) \leq d_1 - t$
- $i = 2 \Rightarrow c_1(t) + c_2(t) \leq d_2 - t$
- $i = 3 \Rightarrow c_1(t) + c_2(t) + c_3(t) \leq d_3 - t$
- $i = 4 \Rightarrow c_1(t) + c_2(t) + c_3(t) + c_4(t) \leq d_4 - t$
An online admission test for EDF

\[ \forall i \sum_{k=1}^{i} c_k(t) \leq d_i - t \]
What next?

Good news!

We want to prove that **EDF is optimal**

(in the sense of feasibility)

Namely:

If a task set is **feasible**, then **EDF** can also generate a feasible schedule for it!
EDF optimality
(in the sense of feasibility)
[Dertouzos 1974]

An algorithm A is **optimal** in the sense of feasibility if it generates a feasible schedule, if there exists one.

How can we prove the optimality of EDF?
EDF optimality
(in the sense of feasibility)
[Dertouzos 1974]

An algorithm A is **optimal** in the sense of feasibility if it generates a feasible schedule, if there exists one.

**Demonstration method (for the proof of EDF)**

It is sufficient to show that, given an arbitrary feasible schedule, the schedule generated by EDF is also feasible.
Dertouzos Transformation

\[ \sigma(t) = \text{task executing at time } t \]
\[ E(t) = \text{task with the minimum } d \text{ at time } t \]
\[ t_E = \text{time at which task } E \text{ is executed} \]

What are \( \sigma(t) \) and \( E(t) \) ?
Dertouzos Transformation

\( \sigma(t) \) = task executing at time \( t \)

\( E(t) \) = task with the minimum \( d \) at time \( t \)

\( t_E \) = time at which task \( E \) is executed

\[\begin{align*}
\sigma(t) &= 4 \\
E(t) &= 2 \\
t_E &= 6
\end{align*}\]

for \((t = 0\) to \(d_{\text{max}} - 1)\)

if \((\sigma(t) \neq E(t))\)

\{ \(\sigma(t_E) = \sigma(t); \sigma(t) = E(t);\) \}

This is a part of \( \sigma \) that is different from EDF
\( \sigma(t) \) = task executing at time \( t \)

\( E(t) \) = task with the minimum \( d \) at time \( t \)

\( t_E \) = time at which task \( E \) is executed

\[
\sigma \neq EDF
\]

\[
\begin{align*}
\text{for } (t = 0 \text{ to } d_{\text{max}} - 1) \quad & \text{if } (\sigma(t) \neq E(t)) \\
& \{ \\
& \quad \sigma(t_E) = \sigma(t); \\
& \quad \sigma(t) = E(t); \\
& \}
\end{align*}
\]
Dertouzos Transformation

\[ \sigma(t) = \text{task executing at time } t \]
\[ E(t) = \text{task with the minimum } d \text{ at time } t \]
\[ t_E = \text{time at which task } E \text{ is executed} \]

\[ \sigma \neq EDF \]
Dertouzos Transformation

\[ \sigma(t) = \text{task executing at time } t \]
\[ E(t) = \text{task with the minimum } d \text{ at time } t \]
\[ t_E = \text{time at which task } E \text{ is executed} \]

\[ \sigma \neq EDF \]

For \( t = 0 \text{ to } d_{max} - 1 \)

if \( \sigma(t) \neq E(t) \)

\{ 
\[ \sigma(t_E) = \sigma(t); \]
\[ \sigma(t) = E(t); \]
\}
Why the Dertouzos transformation preserves schedulability?

- for the advanced slice, this is obvious!
- for the postponed slice, the slack cannot decrease because the deadline of the postponed task MUST have been later than the deadline of the other one (otherwise we wouldn’t swap them)
A property of optimal algorithms

If a task set is not schedulable by an **optimal algorithm**, then it **cannot** be scheduled by **any other algorithm**.

If an algorithm A **minimizes** $L_{max}$ then A is also **optimal** in the sense of feasibility. The opposite is not true.
Agenda

1. Online scheduling algorithms for real-time systems (chapter 3):
   • What do we cover?
     • EDF
     • An interesting yet long “proof of optimality” (from 1974)

2. **Explanation of the Lab assignments**

3. Table-driven scheduling

4. Non-preemptive scheduling
   • Proof of non-optimality of EDF (from 2016)
Your mission

Your mission is online!

Real-Time Systems (IN4343)
- Practical training 2018/2019 -

Your saga begins here

You are hired by a company as an embedded-system engineer whose first task is to improve a legacy embedded system that has been marketed by the company for a couple of decades. This system is kind-of-cursed! Whoever had worked on it had left the company after a week!

Yet here you are! A brave adventurous TUDelft-branded embedded-system engineer who will not only successfully reverse-engineer this system but also improve it by reducing its scheduling overhead. Of course in this battle, your knowledge from the real-time system course is your Friesian horse that moves you forward, your brain is your lance that helps you defeat the problem, and your persistence is your shield that protects you against disappointments! No legacy software beats (or even bites) you as long as you have these three.

Your mission. Break the curse forever! Work on this legacy code and (i) understand it (survive), (ii) improve its efficiency by re-designing how the tasks are scheduled (fight), and (iii) add new features such as faster and more effective scheduling algorithms (win). You will have three assignments for each of these sub-missions. Assignments (i) and (iii) have a one-week due date and assignment (ii) has a two-weeks due date, so prepare yourself for a long battle.

Reward. Your employer will let you pass to the next level to test you with the next monstrous software they have and see if you are worthy of receiving the title of Real-Time System Engineer and perhaps an interesting upgrade in your salary. During your assignments (ii) and (iii), you will receive opportunities to gain optional bonus points that will directly be added to your final grade.

Survival hints. Here are a few tips for your survival during this mission:

1. Keep in mind that you are a real-time system engineer! Use your knowledge and creativity, but beware of your design choices.
2. Be critical and study the code carefully.
3. Write your answers concisely (briefly).
Preparation

Ubuntu 9.10

Simulator for MSP430

Code of the legacy application in C (lab1.zip)

vmware player

Your own operating system (Linux or Windows)

Share folder
Lab 1

• Study the code and find out how the scheduling algorithm and timer interrupt handler work
• Measure overheads (= time that is not spent on executing actual tasks)
• Measure overheads as a function of the number of tasks in the system

\[
\text{Average overhead} = \frac{\text{Total overhead in one hyperperiod}}{\text{length of the hyperperiod}}
\]
coffee time
Agenda

1. Online scheduling algorithms for real-time systems (chapter 3):
   • What do we cover?
     • EDF
     • An interesting yet long “proof of optimality” (from 1974)

2. Explanation of the Lab assignments

3. Table-driven scheduling

4. Non-preemptive scheduling
   • Proof of non-optimality of EDF (from 2016)
Table-driven scheduling

• Store the schedule in a table that is prepared offline
• Dispatch jobs according to the table

Example (for interested students):

Table-driven scheduling in LitmusRT (a Linux-based operating system for real-time systems):
Table-driven scheduling

- Store the schedule in a table that is prepared offline
- Dispatch jobs according to the table

<table>
<thead>
<tr>
<th>Task</th>
<th>Start time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
</tbody>
</table>

Timer interrupt handler:

1. Dispatcher reads the table entry at the current index $i$ and dispatches the task
Table-driven scheduling

- Store the schedule in a table that is prepared offline
- Dispatch jobs according to the table

<table>
<thead>
<tr>
<th>Task</th>
<th>Start time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
</tbody>
</table>

Timer interrupt handler:

1. Dispatcher reads the table entry at the current index $i$ and dispatches the task
2. Increments $i$
Table-driven scheduling

• Store the schedule in a table that is prepared offline
• Dispatch jobs according to the table

<table>
<thead>
<tr>
<th>Task</th>
<th>Start time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
</tbody>
</table>

Timer interrupt handler:

1. Dispatcher reads the table entry at the current index \( i \) and dispatches the task
2. Increments \( i \)
3. Sets the timer interrupt to the “start time” of the current table entry at index \( i \)
Advantages of table-driven scheduling

• **Extremely predictable**
  • Provides full knowledge about when the tasks will be executing
  • Hence, you can analyze the performance of your control system, for example

• **Easy to certify**
  • Avionics industry uses table-driven scheduling

• **Extremely flexible for schedule optimizations**
  • It is easy to include optimization techniques while building the schedule
  • Can handles various system constraints such as precedence constraints, etc.

• **Low runtime overhead**
  • A true O(1) algorithm for scheduling on most hardware platforms

• **Very small “code” footprint**
  • Only requires a few instructions to implement
So do we even need the real-time systems course?!!

Animation: the Incredibles
Disadvantages of table-driven scheduling

• Requires concrete knowledge of task activation times
  • Cannot be applied on event-based systems or dynamic workloads

• Building an optimal schedule might be tractably hard
  • recall: the general scheduling problem is NP-Hard

What else?
It eats up a large amount of memory!

Example:
For a system with 6 periodic tasks, less than 1000 jobs per hyperperiod, and 10 bits to store a table entry, a table can become as big as 10KB.

An Arduino Mega has only 6KB RAM.

Animation: Spirited away (Hayao Miyazaki)
Memory is money!

- Many embedded systems have a limited processing power and memory because
  - memory is expensive
  - consumes a lot of energy
Can we save space by compressing the table?

- We need a loss-less compression
- What should be stored?
  - Task ID
  - Start time
  - Finish time

What time unit?

**Microsecond:** more bits per value but higher time resolution

**Millisecond:** less bits per value, but lower time resolution

Example: $1200\mu s$ -> 1 or 2ms?
Can we save space by compressing the table?

- We need a loss-less compression
- What should be stored?
  - Task ID
  - Start time
  - Finish time

Relative or absolute time values?

Schedule:

<table>
<thead>
<tr>
<th>Absolute times</th>
<th>idle</th>
<th>1</th>
<th>2</th>
<th>idle</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>10</td>
<td>15</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>35</td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many bits per time value? 6 bits

55 < 64 = 2^6
Can we save space by compressing the table?

- We need a loss-less compression
- What should be stored?
  - Task ID
  - Start time
  - Finish time

**relative** or **absolute** time values?

<table>
<thead>
<tr>
<th>Schedule:</th>
<th>idle</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute times</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Relative times</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

**How many bits per time value?**

- 4 bits

15 < 16 = 2^4
Can we save space by compressing the table?

- We need a loss-less compression
- What should be stored?
  - Task ID
  - Start time
  - Finish time

**relative** or **absolute** time values?

**Relative values**: fewer bits, but more prone to clock skews
Can we push it even further?
Offline equivalence technique

It is a technique that allows you to store only a few “crucial” information to rebuild your table at runtime with the help of an online scheduling algorithm.

Paper: RTAS’2017 conference


Outstanding Paper Award
Offline Equivalence

Scan the table and Store differences

Types of irregularities
- Priority inversion
- Idle interval

Modify online scheduler to use differential data

Differential data (irregularities)

The solution

Modified online scheduling algorithm

Online scheduling algorithm

Online policy

Schedule table

Offline table generator
Scan Phase

- **Scan** the table to identify irregularities w.r.t. the online policy and **store** them
  - Priority inversion irregularity
  - Idle interval irregularity

Online policy: Fixed-priority P1<P2<P3

\[
\begin{align*}
\tau_3 &= (8, 60) \\
\tau_2 &= (6, 12) \\
\tau_1 &= (3, 10)
\end{align*}
\]

- The 3rd Job of \( \tau_2 \) starts at 30
  - Only two entries were needed

<table>
<thead>
<tr>
<th>Idle-time irregularity table (IIT)</th>
<th>Priority inversion table (PIT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>From time 9, for 1 time unit</td>
<td>The 3\textsuperscript{rd} Job of ( \tau_2 ) starts at 30</td>
</tr>
</tbody>
</table>
How good is the offline equivalence?

- Implementation platform:
  - Arduino Mega 5056
  - 6 KiB RAM, 256 KiB Flash memory, 16MHz processor speed

- Measured outputs:
  - Required memory for offline equivalence (in Byte)
  - Scheduler’s run time (in microseconds)
Measuring offline equivalence table sizes

6 tasks with time budget 1 minute per task set.
Only those task sets that pass the necessary schedulability test have been considered.
Measuring offline equivalence table sizes

6 tasks with time budget 1 minute per task set.
Only those task sets that pass the necessary schedulability test have been considered.
Lessons learned

• Real-time research is cool, particularly when it solves practical engineering problems

• Think out of the box
Agenda

1. Online scheduling algorithms for real-time systems (chapter 3):
   • What do we cover?
     • EDF
       • An interesting yet long “proof of optimality” (from 1974)

2. Explanation of the Lab assignments

3. Table-driven scheduling

4. Non-preemptive scheduling
   • Proof of non-optimality of EDF (from 2016)
Non-preemptive scheduling

Is this task set feasible?

Yes! It is feasible using a non-work-conserving schedule
Is this task set schedulable by non-preemptive EDF?

NO
Non-preemptive scheduling

Non-preemptive EDF is not optimal for non-preemptive scheduling

There does not exist any optimal work-conserving scheduling policy for non-preemptive tasks! (proof is the previous counter example)
How about “work-conserving” policies?

Is non-preemptive EDF the best choice we have under work-conserving policies?

Formally: is any task set that can be feasibly scheduled by a work-conserving policy, can be also feasibly scheduled by non-preemptive EDF?
Non Preemptive Scheduling

To achieve optimality, an algorithm should be clairvoyant, and decide to leave the CPU idle in the presence of ready tasks:

If we forbid to leave the CPU idle in the presence of ready tasks, then EDF is optimal.

NP-EDF is optimal among non-idle scheduling algorithms

For a long time, researchers believed so

It was only in 2016 that a counter example was found!
EDF is good, but not THAT good!

Non-preemptive EDF (work-conserving)

\[
\begin{align*}
\tau_1 & \quad (C_1, T_1) \\
\tau_2 & \quad (8, 30) \\
\tau_3 & \quad (17, 60) \\
\end{align*}
\]

Lateness = 7

Non-preemptive fixed priority (work-conserving): \( P_1 < P_3 < P_2 \)

ECRTS’2016 conference:
Take-away message

• EDF is a good policy for **preemptive** tasks but not non-preemptive tasks!
• It is **not an optimal** policy among work-conserving scheduling policies for non-preemptive tasks

How good is non-preemptive EDF (NP-EDF) in practice?
Wait for the surprise in our lecture on “non-preemptive scheduling”

Disclaimer

• The course slides came from Koen Langendoen who compiled materials found on the web and obtained from friendly colleagues together.

• Thanks to
  • Giorgio Buttazzo
  • Koen Langendoen
  • Akos Ledeczi
  • Gerd Gross
  • Reinder Bril
  • Zonghua Gu
  • and many others