Schedulability of Sporadic Tasks on Uniprocessor Systems

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

30, April 2014
06, May, 2014
07, May 2014
Recurrent Task Models (Revisited)

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task.

- **Periodic Task** $\tau_i$:
  - A job is released exactly and periodically by a period $T_i$.
  - A phase $\phi_i$ indicates when the first job is released.
  - A relative deadline $D_i$ for each job from task $\tau_i$.
  - $(\phi_i, C_i, T_i, D_i)$ is the specification of periodic task $\tau_i$, where $C_i$ is the worst-case execution time. When $\phi_i$ is omitted, we assume $\phi_i$ is 0.
Recurrent Task Models (Revisited)

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task.

- **Periodic Task** $\tau_i$:
  - A job is released exactly and periodically by a period $T_i$.
  - A phase $\phi_i$ indicates when the first job is released.
  - A relative deadline $D_i$ for each job from task $\tau_i$.
  - $(\phi_i, C_i, T_i, D_i)$ is the specification of periodic task $\tau_i$, where $C_i$ is the worst-case execution time. When $\phi_i$ is omitted, we assume $\phi_i$ is 0.

- **Sporadic Task** $\tau_i$:
  - $T_i$ is the minimal time between any two consecutive job releases.
  - A relative deadline $D_i$ for each job from task $\tau_i$.
  - $(C_i, T_i, D_i)$ is the specification of sporadic task $\tau_i$, where $C_i$ is the worst-case execution time.
Relative Deadline $\iff$ Period (Revisit)

For a task set, we say that the task set is with

- **implicit deadline** when the relative deadline $D_i$ is equal to the period $T_i$, i.e., $D_i = T_i$, for every task $\tau_i$,
- **constrained deadline** when the relative deadline $D_i$ is no more than the period $T_i$, i.e., $D_i \leq T_i$, for every task $\tau_i$, or
- **arbitrary deadline** when the relative deadline $D_i$ could be larger than the period $T_i$ for some task $\tau_i$. 
Some Definitions for Sporadic/Periodic Tasks

- The jobs of task $\tau_i$ are denoted $J_{i,1}, J_{i,2}, \ldots$.
- Periodic Tasks:
  - Synchronous system: Each task has a phase of 0.
  - Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of $T_i$.
- Task utilization of task $\tau_i$: $U_i := \frac{C_i}{T_i}$.
- System (total) utilization: $U(\mathcal{T}) := \sum_{\tau_i \in \mathcal{T}} u_i$. 
Static-Priority Scheduling

- Different jobs of a task are assigned the same priority.
  - $\pi_i$ is the priority of task $\tau_i$.
  - $HP_i$ is the subset of tasks with higher priority than $\tau_i$.
  - Note: we will assume that no two tasks have the same priority. (Why?)
- We will implicitly index tasks in decreasing priority order, i.e., $\tau_i$ has higher priority than $\tau_k$ if $i < k$. 
Static-Priority Scheduling

- Different jobs of a task are assigned the same priority.
  - $\pi_i$ is the priority of task $\tau_i$.
  - $HP_i$ is the subset of tasks with higher priority than $\tau_i$.
  - Note: we will assume that no two tasks have the same priority. (Why?)
- We will implicitly index tasks in decreasing priority order, i.e., $\tau_i$ has higher priority than $\tau_k$ if $i < k$.
- Which strategy is better or the best?
  - largest execution time first?
  - shortest job first?
  - least-utilization first?
  - most importance first?
  - least period first?
Rate-Monotonic (RM) Scheduling (Liu and Layland, 1973)

Priority Definition: A task with a smaller period has higher priority, in which ties are broken arbitrarily.

Example Schedule: \( \tau_1 = (1, 6, 6), \; \tau_2 = (2, 8, 8), \; \tau_3 = (4, 12, 12). \) 
\([(C_i, T_i, D_i)]\)
Liu and Layland (Journal of the ACM, 1973)
Scheduling Algorithms for Multiprogramming in a Hard-Real-Time... by CL Liu - 1973 - Cited by 6917 - Related articles

E. F. Codd, Multiprogram scheduling: parts 3 and 4. scheduling algorithm and external constraints, Communications of the ACM, v.3 n.7, p.413-418, ...

[PDF] Scheduling Algorithms for Multiprogramming in a Hard-Real-Time... File Format: PDF/Adobe Acrobat - Quick View

Liu, C.L. Scheduling algorithms for hard-real-time multiprogramming of a single processor. JPL Space Programs Summary 37-60, Vol. II, Jet Propulsion Lab., ...

citeeex.ist.psu.edu/viewdoc/download?doi=10.1.1.78... - Similar
Deadline-Monotonic (DM) Scheduling (Leung and Whitehead)

Priority Definition: A task with a smaller relative deadline has higher priority, in which ties are broken arbitrarily.

Example Schedule: $\tau_1 = (2, 8, 4)$, $\tau_2 = (1, 6, 6)$, $\tau_3 = (4, 12, 12)$. $[(C_i, T_i, D_i)]$
Optimality (or not) of RM and DM

Example Schedule: \( \tau_1 = (2, 4, 4), \tau_2 = (5, 10, 10) \)

The above system is schedulable.
Optimality (or not) of RM and DM

Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$

The above system is schedulable.

No static-priority scheme is optimal for scheduling periodic tasks: However, a deadline will be missed, regardless of how we choose to (statically) prioritize $\tau_1$ and $\tau_2$.

Corollary

Neither RM nor DM is optimal.
Aside: Critical Instants

**Definition**

The critical instant of a task \( \tau_i \) is a time instant such that:

1. the job of \( \tau_i \) released at this instant has the maximum response time of all jobs in \( \tau_i \), if the response time of every job of \( \tau_i \) is at most \( T_i \), the period of \( \tau_i \), and

2. the response time of the job released at this instant is greater than \( T_i \) if the response time of some job in \( \tau_i \) exceeds \( T_i \).

Informally, a critical instant of \( \tau_i \) represents a worst-case scenario from \( \tau_i \)’s standpoint.
Critical Instants in Static-Priority Systems

Theorem

[Liu and Layland, JACM 1973] The critical instance of task $\tau_i$ for a set of independent, preemptable periodic tasks with relative deadlines equal to their respective periods is to release the first jobs of all the higher-priority tasks at the same time.

*We are not saying that $\tau_1, \ldots, \tau_i$ will all necessarily release their first jobs at the same time, but if this does happen, we are claiming that the time of release will be a critical instant for task $\tau_i$.**
Shifting the release time of tasks together will increase the response time of task $\tau_i$.

- Consider a job of $\tau_i$, released at time $t'$, with completion time $t_R$.
- Let $t_{-1}$ be the latest *idle instant* for $\tau_1, \ldots, \tau_{i-1}$ at or before $t_R$.
- Let $J$ be $\tau_i$'s job released at $t'$. 
We will show that shifting the release time of tasks together will increase the response time of task $\tau_i$.  
- Moving $J$ from $t'$ to $t_{-1}$ does not decrease the completion time of $J$.  

We will show that shifting the release time of tasks together will increase the response time of task $\tau_i$.

- Releasing $\tau_1$ at $t_{-1}$ does not decrease the completion time of $J$. 
We will show that shifting the release time of tasks together will increase the response time of task $\tau_i$.

- Releasing $\tau_2$ at $t_{-1}$ does not decrease the completion time of $J$.
- Repeating the above movement and proves the criticality of the critical instant
Outline

Schedulability for Static-Priority Scheduling
  Demand-Based Analysis
  Optimality of RM
  Utilization-Based Analysis (Relative Deadline = Period)
  Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling
Necessary/Sufficient Schedulability Test

- The issue for timing analysis is on how to analyze the schedulability.
  - Sufficient Test: If A holds, then the task set is schedulable (by EDF, RM, or DM).
  - Necessary Test: If the task set is schedulable by EDF (or RM/DM), then B holds.
  - Exact Test: The task set is schedulable by EDF (or RM/DM) if and only if $A^*$ holds.
Necessary and Sufficient (Exact) RM-Schedulability

- TDA can be applied to produce a schedulability test for any fixed-priority algorithm that ensures that each job of every task completes before the next job of that task is released.

- For some important task models and scheduling algorithms, this schedulability test will be necessary and sufficient.
Schedulability Condition

According to the critical instant theorem, to test the schedulability of task $\tau_i$, we have to

1. release all the higher-priority tasks at time 0 together with task $\tau_i$

2. release all the higher-priority task instances as early as they can

We can simply simulate the above behavior to verify whether task $\tau_i$ misses the deadline.

Several examples are used in Slide 6/8/9 to demonstrate this behavior.
Schedulability Test

The time-demand function $W_i(t)$ of the task $\tau_i$ is defined as follows:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{t}{T_j} \right\rfloor C_j.$$

**Theorem**

A system $\mathcal{T}$ of periodic, independent, preemptable tasks is schedulable on one processor by algorithm A if

$$\forall \tau_i \in \mathcal{T} \exists t \text{ with } 0 < t \leq D_i \text{ and } W_i(t) \leq t$$

holds. This condition is also necessary for synchronous, periodic task systems and also sporadic task sets.

Note that this holds for implicit-deadline and constrained-deadline task sets. The sufficient condition can be proved by contradiction.
How to Use TDA?

The theorem of TDA might look strong as it requires to check all the time $t$ with $0 < t \leq D_i$ for a given $\tau_i$. There are two ways to avoid this:

- Iterate using $t(k + 1) := W_i(t(k))$, starting with $t(0) := \sum_{i=1}^{j} C_j$, and stopping when, for some $\ell$, $t(\ell) = W_i(t(\ell))$ or $t(\ell) > D_i$.

- Only consider $t \in \{\ell T_j - \epsilon \mid 1 \leq j \leq i, \ell \in \mathcal{N}^+\}$, where $\epsilon$ is a constant close to 0. That is, only consider $t$ at which a job of higher-priority tasks arrives.
For analyzing whether task $\tau_i$ can meet the timing constraint, the complexity is $O(iD_i)$.

- It is polynomial time if the input is in the unary format. That is, when $D_i$ is 6, the input is 111111 instead of 110 in the binary format.
- It is exponential time in the binary format.
- Formally, this is called with pseudo-polynomial time complexity.

**Theorem**

Eisenbrand and Rothvoss [RTSS 2008]: Fixed-Priority Real-Time Scheduling: Response Time Computation Is $\mathcal{NP}$-Hard
Theorem

A system \( T \) of independent, preemptable, synchronous periodic tasks that have relative deadlines equal to their respective periods can be feasibly scheduled on one processor according to the RM algorithm whenever it can be feasibly scheduled according to any static priority algorithm.

We will only discuss systems with 2 tasks, and the generalization is left as an exercise.

- Suppose that \( T_1 = D_1 < D_2 = T_2 \) and \( \tau_2 \) is with higher priority.
- We would like to swap the priorities of \( \tau_1 \) and \( \tau_2 \).
- Without loss of generality, the response time of \( \tau_1 \) after priority swapping is always equal to (or no more than) \( C_1 \).
- By the critical instant theorem, we only need to check response time of the first job of \( \tau_2 \) during a critical instant.
- Assuming that non-RM priority ordering is schedulable, the critical instant theorem also implies that \( C_1 + C_2 \leq T_1 \).
After swapping ($\tau_1$ has higher priority), there are two cases:

**Case 1**

There is sufficient time to complete all $F$ jobs of $\tau_1$ before the second job arrival of $\tau_2$, where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$. In other words, $C_1 + F \cdot T_1 < T_2$.

To be schedulable

$(F + 1)C_1 + C_2 \leq T_2$ must hold.

By $C_1 + C_2 \leq T_1$, we have

\[ F(C_1 + C_2) \leq F \cdot T_1 \]

$F \geq 1 \Rightarrow FC_1 + C_2 \leq F \cdot T_1$

\[ (F + 1)C_1 + C_2 \leq F \cdot T_1 + C_1 \]

\[ \Rightarrow (F + 1)C_1 + C_2 < T_2 \]
After swapping ($\tau_1$ has higher priority), there are two cases:

**Case 2**

The $F$-th job of $\tau_1$ does not complete before the arrival of the second job of $\tau_2$. In other words, $C_1 + F \cdot T_1 \geq T_2$, where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$.

By $C_1 + C_2 \leq T_1$, we have

$$F(C_1 + C_2) \leq F \cdot T_1$$

$$F \geq 1 \Rightarrow FC_1 + C_2 \leq F \cdot T_1 \leq T_2$$

To be schedulable

$FC_1 + C_2 \leq FT_1$ must hold.
Remarks on the Optimality

We have shown that if any two-task system with implicit deadlines \((D_i = T_i)\) is schedulable according to arbitrary fixed-priority assignment, then it is also schedulable according to RM.

**Exercise:** Complete proof by extending argument to \(n\) periodic tasks.

**Note:** When \(D_i \leq T_i\) for all tasks, DM (Deadline Monotonic) can be shown to be an optimal static-priority algorithm using similar argument. The proof is also left as an exercise.
Outline

Schedulability for Static-Priority Scheduling
  Demand-Based Analysis
  Optimality of RM
  Utilization-Based Analysis (Relative Deadline = Period)
  Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling
Definitions

- Task utilization:
  \[ U_i := \frac{C_i}{T_i} \]

- System (total) utilization:
  \[ U(T) := \sum_{\tau_i \in T} \frac{C_i}{T_i} \]
Harmonic Real-Time Systems

Definition

A system of periodic tasks is said with harmonic periods (also: *simply periodic*) if for every pair of tasks $\tau_i$ and $\tau_k$ in the system where $T_i < T_k$, $T_k$ is an integer multiple of $T_i$.

For example: Periods are 2, 6, 12, 24.

Theorem

[Kuo and Mok]: A system $\mathcal{T}$ of harmonic, independent, preemptable, and implicit-deadline tasks is schedulable on one processor according to the RM algorithm if and only if its total utilization $U(\mathcal{T}) = \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j}$ is less than or equal to 1.
Proof for Harmonic Systems

The case for the “only-if” part is skipped.

By using the contrapositive proof approach, suppose that $T$ is not schedulable and $\tau_i$ misses its deadline. We will prove that the utilization must be larger than 1.

- The response time of $\tau_i$ is larger than $D_i$.
- By critical instants, releasing all the tasks $\tau_1, \tau_2, \ldots, \tau_i$ at time 0 will lead to a response time of $\tau_i$ larger than $D_i$. 
As the schedule is workload-conserving, we know that from time 0 to time $D_i$, the whole system is executing jobs. Therefore,

$$D_i < \text{the workload released in time interval } [0, D_i)$$

$$= \sum_{j=1}^{i} C_j \cdot \left( \text{the number of job releases of } \tau_j \text{ in time interval } [0, D_i) \right)$$

$$= \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{D_i}{T_j} \right\rceil =^* \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},$$

where $=^*$ is because $D_i = T_i$ is an integer multiple of $T_j$ when $j \leq i$. 

Proof for Harmonic Systems (cont.)
Proof for Harmonic Systems (cont.)

As the schedule is workload-conserving, we know that from time 0 to time $D_i$, the whole system is executing jobs. Therefore,

$$D_i < \text{the workload released in time interval } [0, D_i)$$

$$= \sum_{j=1}^{i} C_j \cdot (\text{the number of job releases of } \tau_j \text{ in time interval } [0, D_i))$$

$$= \sum_{j=1}^{i} C_j \cdot \left\lfloor \frac{D_i}{T_j} \right\rfloor = * \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},$$

where $=*$ is because $D_i = T_i$ is an integer multiple of $T_j$ when $j \leq i$.

By canceling $D_i$, we reach the contradiction by having

$$1 < \sum_{j=1}^{i} \frac{C_j}{T_j} \leq \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j}.$$
Utilization-Based Schedulability Test

- Task utilization:
  \[ u_i := \frac{C_i}{T_i}. \]

- System (total) utilization:
  \[ U(\mathcal{T}) := \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{T_i}. \]

A task system \( \mathcal{T} \) fully utilizes the processor under scheduling algorithm \( A \) if any increase in execution time (of any task) causes \( A \) to miss a deadline. In this case, \( U(\mathcal{T}) \) is an upper bound on utilization for \( A \), denoted \( U_{ub}(\mathcal{T}, A) \).

\( U_{lub}(A) \) is the least upper bound for algorithm \( A \):
\[ U_{lub}(A) = \min_{\mathcal{T}} U_{ub}(\mathcal{T}, A) \]
What is $U_{lb}(A)$ for?

![Diagram showing feasible, unsure, and infeasible regions for $U_{lb}(A)$ for different $T_i$]
Liu and Layland Bound

**Theorem**

[Liu and Layland] A set of \( n \) independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization \( U \) is at most \( n(2^{\frac{1}{n}} - 1) \). In other words,

\[
U_{lub}(RM, n) = n(2^{\frac{1}{n}} - 1) \geq 0.693.
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( U_{lub}(RM, n) )</th>
<th>( n )</th>
<th>( U_{lub}(RM, n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.828</td>
<td>3</td>
<td>0.779</td>
</tr>
<tr>
<td>4</td>
<td>0.756</td>
<td>5</td>
<td>0.743</td>
</tr>
<tr>
<td>6</td>
<td>0.734</td>
<td>7</td>
<td>0.728</td>
</tr>
<tr>
<td>8</td>
<td>0.724</td>
<td>9</td>
<td>0.720</td>
</tr>
<tr>
<td>10</td>
<td>0.717</td>
<td>( \infty )</td>
<td>0.693 = ln2</td>
</tr>
</tbody>
</table>
Least Upper Bound

RM Bound

Least Upper Bound

n

10  20  30  40  50  60  70  80  90  100
Proof Sketch for $U_{ulb}(RM, n)$

Note: The original proof for this theorem by Liu and Layland is not correct. For a corrected proof, see R. Devillers & J. Goossens at http://dev.ulb.ac.be/sched/articles/lub.ps. Note the proof presented here is VERY different from the others, including the one from Buttazzo’s textbook.

Without loss of generality, we will only consider task sets with distinct periods, i.e., $T_1 < T_2 < \cdots < T_n$.

1. We will start from the exact test and analyze the schedulability under RM of task $\tau_n$.
2. This will easily lead us to consider only the special case where $T_n \leq 2T_1$.
3. We will then show the schedulability condition of $\tau_n$ under RM.
4. The least utilization bound is then derived based on the above schedulability condition.
Utilization Bound Proof: Step 1

**Theorem**

[Bini and Buttazzo, ECRTS 2001] A system of n independent, pre-emptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if

\[ \prod_{i=1}^{n} (U_i + 1) \leq 2. \]

Suppose that task \( \tau_n \) is not schedulable under RM. We will prove that \( \prod_{i=1}^{n} (U_i + 1) > 2. \)
Worst-Case: \( T_n \leq 2 T_1 \)

By the exact test, we know that for all \( 0 < t \leq T_n \)

\[
W_n(t) = C_n + \sum_{i=1}^{n-1} \left\lceil \frac{t}{T_i} \right\rceil C_i > t.
\]

Now, suppose again \( F_i \) is \( \left\lfloor \frac{T_n}{T_i} \right\rfloor \). For all \( 0 < t \leq T_n \)

\[
\left\lfloor \frac{t}{T_i} \right\rfloor C_i \leq \left\lfloor \frac{t}{F_i T_i} \right\rfloor F_i C_i
\]

Therefore, by changing the period of task \( \tau_i \) to \( F_i T_i \) and the execution time from \( C_i \) to \( F_i C_i \), task \( \tau_n \) in the new task set remains unschedulable under RM.

After changing the periods, we reorder the tasks according to their new periods. Does this affect the non-schedulability of task \( \tau_n \)?
\[ T_n \leq 2T_1 \text{ (cont.)} \]
Hyperbolic Bound: Structure

For the rest of the proof, we only consider $T_n \leq 2T_1$. The non-schedulability also implies the following structure:

$$C_n + \sum_{j=1}^{n-1} C_j + \sum_{j=0}^{i-1} C_j > T_i, \forall i = 1, 2, \ldots, n - 1,$$

$$C_n + 2 \sum_{j=1}^{n-1} C_j > T_n,$$

where $C_0$ is defined as 0 for brevity.
Hyperbolic Bound: Structure

That is, $C_n$ must be \textit{sufficiently large} to enforce the above conditions.

Let’s now recall what we were doing:

- We were given a set of tasks, in which the utilization $U_i$ of each task $\tau_i$ is given.
- We wanted to prove that the task set is always schedulable under RM no matter how the periods are assigned under certain utilization constraints.
- What we have done so far is the contraposition that when the utilization above certain conditions, there exists at least one assignment of periods to make task $\tau_n$ not schedulable under RM.
Hyperbolic Bound: $U_n$

For a critical value of $U_n$, if we reduce $U_n$ by a small value, the above non-schedulability condition will not be satisfied any more. So, the critical value is equivalent to the minimum $U_n$ to enforce the following condition:

$$C_n + \sum_{j=1}^{n-1} C_j + \sum_{j=0}^{i-1} C_j \geq T_i, \forall i = 1, 2, \ldots, n - 1,$$

$$C_n + 2 \sum_{j=1}^{n-1} C_j = T_n,$$

In fact, we can also normalize $T_n$ to 1. The above condition to get the minimum $U_n$ is equivalent to the following linear programming:

$$\text{minimize } C_n = T_n - 2 \sum_{j=1}^{n-1} U_j T_j$$

$$\text{s.t. } T_n - 2 \sum_{j=1}^{n-1} U_j T_j + \sum_{j=1}^{n-1} U_j T_j + \sum_{j=0}^{i-1} U_j T_j \geq T_i, \forall i = 1, \ldots, n - 1$$
Extreme Point Theory in Linear Programming

minimize $C_n = T_n - 2 \sum_{j=1}^{n-1} U_j T_j$

s.t. $T_i = T_n - \sum_{j=i}^{n-1} U_j T_j \geq T_i, \forall i = 1, \ldots, n - 1$

The optimal solution of the above linear programming is achieved when all the $n - 1$ linear constraints are with $=$ instead of $\geq$ by the extreme point theory. That is, the minimum $U_n$ is achieved when

$$T_i = T_n - \sum_{j=i}^{n-1} U_j T_j, \forall i = 1, \ldots, n - 1$$

$$C_i = T_{i+1} - T_i, \forall i = 1, \ldots, n - 1$$
Hyperbolic Bound: Final

\[ U_i = \frac{C_i}{T_i} = \frac{T_{i+1} - T_i}{T_i} = \frac{T_{i+1}}{T_i} - 1, \forall i = 1, \ldots, n - 1 \]

\[ C_n^* = 2T_1 - T_n \Rightarrow U_n^* = 2 \frac{T_1}{T_n} - 1, \]

where \( C_n^* \) is the optimal solution of the above linear programming.

The non-schedulability of task \( \tau_n \) implies that

\[ U_n > U_n^* = 2 \frac{T_1}{T_n} - 1 = 2 \left( \frac{T_1}{T_2} \frac{T_2}{T_3} \ldots \frac{T_{n-1}}{T_n} \right) - 1 \]

\[ = 2 \frac{1}{\prod_{i=1}^{n-1} (U_i + 1)} - 1 \]

\[ \Rightarrow \prod_{i=1}^{n} (U_i + 1) > 2. \]
Theorem

[Bini and Buttazzo, ECRTS 2001] A system of \( n \) independent, pre-emptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if

\[
\prod_{i=1}^{n}(U_i + 1) \leq 2.
\]
Utilization Bound Proof: Step 2 Calculate $U_{lub}(RM, n)$

- So what is the maximum $\sum_{i=1}^{n} U_i$ to enforce $\Pi_{i=1}^{n}(U_i + 1) \leq 2$?
  - It should be clear that the maximum $\sum_{i=1}^{n} U_i$ happens when $\Pi_{i=1}^{n}(U_i + 1) = 2$, and $U_i = 2^{\frac{1}{n}} - 1$.
  - $U_{lub}(RM, n) = n(2^{\frac{1}{n}} - 1) \geq \lim_{n \to \infty} n(2^{\frac{1}{n}} - 1) = ln2$.

This concludes the proof of the following theorem:

**Theorem**

[Liu and Layland, JACM 1973] A set of $n$ independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization $U$ is at most $n(2^{\frac{1}{n}} - 1)$. In other words,

$$U_{lub}(RM, n) = n(2^{\frac{1}{n}} - 1) \geq 0.693.$$
Least Upper Bound

The graph illustrates the relationship between $U_1$ and $U_2$. The solid red line represents the hyperbolic analysis, while the dotted green line represents the least upper bound. The x-axis represents $U_1$ ranging from 0 to 1, and the y-axis represents $U_2$ ranging from 0 to 1.
On-Site Exercise: Is This Task Set Schedulable?

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>0.2</th>
<th>2</th>
<th>2</th>
<th>1.5</th>
<th>1</th>
<th>14</th>
<th>28.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>2</td>
<td>7</td>
<td>14</td>
<td>26</td>
<td>26</td>
<td>79</td>
<td>292</td>
</tr>
<tr>
<td>$D_i$</td>
<td>2</td>
<td>6</td>
<td>13</td>
<td>25</td>
<td>26</td>
<td>77</td>
<td>291</td>
</tr>
</tbody>
</table>
On-Site Exercise: Is This Task Set Schedulable?

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>0.2</th>
<th>2</th>
<th>2</th>
<th>1.5</th>
<th>1</th>
<th>14</th>
<th>28.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>2</td>
<td>7</td>
<td>14</td>
<td>26</td>
<td>26</td>
<td>79</td>
<td>292</td>
</tr>
<tr>
<td>$D_i$</td>
<td>2</td>
<td>6</td>
<td>13</td>
<td>25</td>
<td>26</td>
<td>77</td>
<td>291</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>0.2</th>
<th>2</th>
<th>2</th>
<th>1.5</th>
<th>1</th>
<th>14</th>
<th>28.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>2</td>
<td>6</td>
<td>13</td>
<td>25</td>
<td>26</td>
<td>77</td>
<td>291</td>
</tr>
<tr>
<td>$D_i$</td>
<td>2</td>
<td>6</td>
<td>13</td>
<td>25</td>
<td>26</td>
<td>77</td>
<td>291</td>
</tr>
</tbody>
</table>
On-Site Exercise: Is This Task Set Schedulable?

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>0.2</th>
<th>2</th>
<th>2</th>
<th>1.5</th>
<th>1</th>
<th>14</th>
<th>28.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>2</td>
<td>7</td>
<td>14</td>
<td>26</td>
<td>26</td>
<td>79</td>
<td>292</td>
<td></td>
</tr>
<tr>
<td>$D_i$</td>
<td>2</td>
<td>6</td>
<td>13</td>
<td>25</td>
<td>26</td>
<td>77</td>
<td>291</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>0.2</th>
<th>2</th>
<th>2</th>
<th>1.5</th>
<th>1</th>
<th>14</th>
<th>28.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$′</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>24</td>
<td>72</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>$D_i$′</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>24</td>
<td>72</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>$U_i$′</td>
<td>0.1</td>
<td>1/3</td>
<td>1/6</td>
<td>0.0625</td>
<td>0.0417</td>
<td>0.195</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>
Remarks on Harmonic Task Set

- Now, we know that if the total utilization is larger than 0.693, the utilization-bound schedulability cannot provide guarantees for schedulability or unschedulability.
- Sometimes, we can manipulate the periods such that the new task set is a harmonic task set and its schedulability can be used.
Outline

Schedulability for Static-Priority Scheduling
  Demand-Based Analysis
  Optimality of RM
  Utilization-Based Analysis (Relative Deadline = Period)
  Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling
TDA for Arbitrary Deadlines (Details are in Appendix)

- The TDA scheduling condition is valid only if each job of every task completes before the next job of that task is released.
- We now consider a schedulability check in which tasks may have *relative deadlines larger than their periods*.
  - Note: In this model, a task may have multiple ready jobs. We assume they are scheduled on a First-Come-First Serve (FCFS) basis.
The worst-case response time of $\tau_i$ by only considering the first job of $\tau_i$ at the critical instant is too optimistic when the relative deadline of $\tau_i$ is larger than the period.

Consider two tasks:

- $\tau_1$ has period 70 and execution time 26 and $\tau_2$ is with period 100 and execution time 62.
- $\tau_2$’s seven jobs have the following response times, respectively: 114, 102, 116, 104, 118, 106, 94.
- Note that the first job’s response time is not the longest.
Priority Ordering for Tasks with Arbitrary-Deadline

- There is no greedy strategy for optimal ordering
  - DM or RM is not an optimal static-priority scheme any more.
- Audsley’s approach (1991):
  - Use TDA to find the worst-case response time of task $\tau_i$ by assuming the others have higher priority
  - Among those tasks whose worst-case response times are less than or equal to the relative deadlines, choose one of them as the lowest-priority task
  - Reduce the problem by removing this lowest-priority task and repeat the above procedure
- Audsley’s approach is optimal.
Outline

Schedulability for Static-Priority Scheduling
  Demand-Based Analysis
  Optimality of RM
  Utilization-Based Analysis (Relative Deadline = Period)
  Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling
Utilization-Based Test for EDF Scheduling

**Theorem**

Liu and Layland: A task set $\mathcal{T}$ of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization $U$ is at most one.

**Proof**

- **The only if part** is obvious: If $U > 1$, then some task clearly must miss a deadline. So, we concentrate on the if part.
- **We prove the contrapositive**, i.e., if $\mathcal{T}$ is not schedulable, then $U > 1$.
  - Let $J_{i,k}$ be the first job to miss its deadline.
  - Let $t_{-1}$ be the last idle instant before $d_{i,k}$, the absolute deadline of $J_{i,k}$.
  - $t_{-1}$ could be 0 if there is no idle time.

(cont.)
Proof of Utilization-Bound Test for EDF

Proof.

Because $J_{i,k}$ missed its deadline, we know that

$$d_{i,k} - t_{-1} < \text{demand in } [t_{-1}, d_{i,k})$$

by jobs with absolute deadline no more than $d_{i,k}$

$$= \sum_{j=1}^{n} \left\lfloor \frac{d_{i,k} - t_{-1}}{T_j} \right\rfloor C_j$$

$$\leq \sum_{j=1}^{n} \frac{d_{i,k} - t_{-1}}{T_j} C_j$$

By cancelling $d_{i,k} - t_{-1}$, we conclude the proof by

$$1 < \sum_{j=1}^{n} \frac{C_j}{T_j} = U.$$
Proof of Utilization-Bound Test for EDF

Proof.

Because \( J_{i,k} \) missed its deadline, we know that

\[
d_{i,k} - t_{-1} < \text{demand in } [t_{-1}, d_{i,k})
\]

by jobs with absolute deadline no more than \( d_{i,k} \)

\[
= \sum_{j=1}^{n} \left[ \frac{d_{i,k} - t_{-1}}{T_j} \right] C_j
\]

\[
\leq \sum_{j=1}^{n} \frac{d_{i,k} - t_{-1}}{T_j} C_j
\]

By cancelling \( d_{i,k} - t_{-1} \), we conclude the proof by

\[
1 < \sum_{j=1}^{n} \frac{C_j}{T_j} = U.
\]
Relative Deadlines Less than Periods

Theorem

A task set \( \mathcal{T} \) of independent, preemptable, periodic tasks with relative deadlines equal to or less than their periods can be feasibly scheduled (under EDF) on one processor if

\[
\sum_{k=1}^{n} \frac{C_k}{\min\{D_k, T_k\}} \leq 1.
\]

Note: This theorem only gives sufficient condition.
Necessary and Sufficient Conditions

**Theorem**

Define demand bound function \( dbf(\tau_i, t) \) as

\[
\text{dbf}(\tau_i, t) = \max \left\{ 0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \right\}
\]

\[
C_i = \max \left\{ 0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right\}
\]

A task set \( \mathcal{T} \) of independent, preemptable, periodic tasks can be feasibly scheduled (under EDF) on one processor if and only if \( \forall L \geq 0, \sum_{i=1}^{n} \text{dbf}(\tau_i, L) \leq L \).
Proof for EDF Schedulability Test

- The processor demand in time interval \([t_1, t_2]\) is the demand that must be finished in interval \([t_1, t_2]\). That is, only jobs that arrive no earlier than \(t_1\) and have absolute deadline no more than \(t_2\) are considered.

- The processor demand \(g_i([t_1, t_2])\) contributed by task \(\tau_i\) is

\[
g_i([t_1, t_2]) = C_i \cdot \max \left\{ 0, \left\lfloor \frac{t_2 + T_i - D_i - \phi_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \phi_i}{T_i} \right\rceil \right\}
\]

- The feasibility is guaranteed if and only if in any interval \([t_1, t_2]\), the processor demand is no more than the available time, i.e.,

\[
t_2 - t_1 \geq \sum_{i=1}^{n} g_i(t_1, t_2) \geq \sum_{i=1}^{n} \left\lfloor \frac{t_2 + T_i - D_i - t_1}{T_i} \right\rfloor
\]

- Replacing \(t_2 - t_1\) by \(L\), we conclude the proof.
Complexity of the Exact Analysis

For analyzing whether a task set can be schedulable by EDF, the time complexity is $O(nL_{\text{max}})$, where $L_{\text{max}}$ is the hyper-period $\text{LCM}(T_1, T_2, \ldots, T_n)$.

- It takes pseudo-polynomial time.

**Theorem**

Eisenbrand and Rothvoss [SODA 2010]: testing EDF schedulability of such a task set is (weakly) co-$NP$-hard. That is, deciding whether a task set is not schedulable by EDF is (weakly) $NP$-hard.
Comparison between RM and EDF (Implicit Deadlines)

<table>
<thead>
<tr>
<th><strong>RM</strong></th>
<th><strong>EDF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Low run-time overhead: $O(1)$ with priority sorting in advance</td>
<td>• High run-time overhead: $O(\log n)$ with balanced binary tree</td>
</tr>
<tr>
<td>• Optimal for static-priority</td>
<td>• Optimal for dynamic-priority</td>
</tr>
<tr>
<td>• Schedulability test is $NP$-hard (even if the relative deadline = period)</td>
<td>• Schedulability test is easy (when the relative deadline = period)</td>
</tr>
<tr>
<td>• Least upper bound: 0.693</td>
<td>• Least upper bound: 1</td>
</tr>
<tr>
<td>• In general, more preemption</td>
<td>• In general, less preemption</td>
</tr>
</tbody>
</table>
Appendix: TDA for Arbitrary Deadlines
Busy Intervals

Definition

A $\tau_i$-level busy interval $(t_0, t]$ of task $\tau_i$ begins at an instant $t_0$ when

1. all jobs in $\tau_i$ released before $t$ have completed, and
2. a job of $\tau_i$ releases.

The interval ends at the first instant $t$ after $t_0$ when all jobs in $\tau_i$ released since $t_0$ are complete.
Theorem

We are given $\mathcal{T}$ of sporadic, independent, preemptable tasks.

1. If $\forall \tau_i \in \mathcal{T} \exists t$ with $0 < t \leq \min\{T_i, D_i\}$ and $W_i(t) \leq t$, then $\mathcal{T}$ is schedulable on one processor by algorithm A for priority ordering.

2. Otherwise, solving the following equation iteratively

$$t^{(k+1)} = \sum_{j=1}^{i} \left\lfloor \frac{t^{(k)}}{T_j} \right\rfloor C_j,$$

with initialization $t^{(0)} = \sum_{j=1}^{i} C_j$. If the maximum response time of the jobs of $\tau_i$ released in time $(0, t]$ is less than the relative deadline, $\mathcal{T}$ is schedulable; otherwise $\mathcal{T}$ is not schedulable.
Response Times

**Lemma**

The maximum response time $W_{i,j}$ of the $j$-th job of $\tau_i$ in an in-phase $\tau_i$ busy period is equal to the smallest value of $t$ that satisfies the equation

$$t = w_{i,j}(t + (j - 1) \cdot T_i) - (j - 1) \cdot T_i,$$

where $w_{i,j}(t) = jC_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$.

This should be clear now.
An Example TDA Analysis

Suppose that $D_2$ is 120 for this example.

\[
t = w_{2,1}(t) = C_2 + \sum_{j=1}^{1} \left\lfloor \frac{t}{T_j} \right\rfloor C_j = 62 + \left\lfloor \frac{t}{70} \right\rfloor 26 \rightarrow W_{2,1} = 114
\]

\[
t = w_{2,2}(t) = 124 + \left\lfloor \frac{t + 100}{70} \right\rfloor 26 - 100 \rightarrow W_{2,2} = 102
\]

\[
t = w_{2,3}(t) = 186 + \left\lfloor \frac{t + 200}{70} \right\rfloor 26 - 200 \rightarrow W_{2,3} = 116
\]
Correctness of the TDA for Arbitrary Relative Deadlines

**Lemma**

The response time $W_{i,j}$ of the $j$-th job of $\tau_i$ executed in an in-phase $\tau_i$ busy interval is no less than the response time of the $j$-th job of $\tau_i$ executed in any $\tau_i$ busy interval.

**Lemma**

The number of jobs in $\tau_i$ that are executed in an in-phase $\tau_i$ busy interval is never less than the number of jobs in this task that are executed in a $\tau_i$ busy interval of arbitrary phase.