Schedulability of Periodic and Sporadic Task Sets on Uniprocessor Systems

Jan Reineke

Saarland University

July 16, 2015
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.
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 vs Period (revisited)

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: $U(T) := \sum_{\tau_i \in T} u_i$. 
Outline

1 Schedulability Analysis for Static-Priority Scheduling
   - Utilization-Based Analysis (Relative Deadline = Period)
   - Demand-Based Analysis

2 Schedulability Analysis for Dynamic-Priority Scheduling
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.

- 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?
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.

- 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)
Liu and Layland (Journal of the ACM, 1973)

Scholarly articles for Scheduling algorithms for multiprogramming
Scheduling algorithms for multiprogramming in a hard- ... - Liu - Cited by 6917
scheduling algorithms - Liu - Cited by 98
... scheduling algorithms for a model of multiprogramming ... - Krause - Cited by 57

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, ...
portal.acm.org/citation.cfm?id=321743 - Similar

[PDF] Scheduling Algorithms for Multiprogramming in a Hard- Real-Time ...
File Format: PDF/Adobe Acrobat - Quick View
by CL LIU - 1973 - Cited by 6917 - Related articles
Liu, C.L. Scheduling algorithms for hard-real-time multiprogramming of a single pro-
cessor. JPL Space Programs Summary 37-60, Vol. II, Jet Propulsion Lab., ...
citeseerx.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.
No static-priority scheme is optimal for scheduling periodic tasks: 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.
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: 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

A 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 $D_i$, the relative deadline of $\tau_i$, and

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

Informally, a critical instant of $\tau_i$ represents a worst-case scenario from $\tau_i$'s standpoint.
Theorem

[Liu and Layland, JACM 1973] A critical instant 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 $\tau_i$. 
Critical Instants: Informal Proof

We will show that 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 $J$. 
Critical Instants: Informal Proof

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 $J$. 
Critical Instants: Informal Proof

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
Harmonic Real-Time Systems

Definition
A system of periodic tasks harmonic (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 $U = \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j}$ is less than or equal to one.
Proof for Harmonic Systems ("if" part)

The case for the "only-if" part is similar and left as an exercise.

Suppose for a contradiction that $T$ is not schedulable and that $\tau_i$ misses its deadline.

- 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 $D_i$. 
Proof for Harmonic Systems (cont.)

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

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

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

$$= \sum_{j=1}^{i} C_j \cdot \left[ \frac{D_i}{T_j} \right] = \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},$$

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

$$1 < \sum_{j=1}^{i} \frac{C_j}{T_j} \leq \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j} \leq 1.$$
Proof for Harmonic Systems (cont.)

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

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

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

$$= \sum_{j=1}^{i} C_j \cdot \left[ \frac{D_i}{T_j} \right] = \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},$$

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

$$1 < \sum_{j=1}^{i} \frac{C_j}{T_j} \leq \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j} \leq 1.$$
Optimality Among Static-Priority Algorithms

Theorem

A system $\mathcal{T}$ 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 $C_1 + C_2 \leq T_1$. 
Optimality Among Static-Priority Algorithms (cont.)

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 $F = \left\lceil \frac{T_2}{T_1} \right\rceil$. In other words, $C_1 + F \cdot T_1 < T_2$.

![Diagram showing the timeline for Case 1]

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 $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$.

To be schedulable

$FC_1 + C_2 \leq FT_1$ must hold.

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

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

$F \geq 1 \Rightarrow FC_1 + C_2 \leq F \cdot T_1$
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. Proof left as an exercise.
Utilization-Based Schedulability Test

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

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

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.

\( UB(A) \) is the utilization bound for algorithm \( A \):

\[ UB(A) := \bigcup \{ U \in \mathbb{R} \mid \forall \mathcal{T}. U(\mathcal{T}) \leq U \Rightarrow \mathcal{T} \text{ is schedulable under } A \} \]
\[ = \bigcap \{ U(\mathcal{T}) \mid \mathcal{T} \text{ fully utilizes the processor under } A \} \]
What is $UB(A)$ useful for?

- $U(T_1)$
- $U(T_2)$
- $U(T_3)$
- $U(T_4)$
- $U(T_5)$
- $U(T_?)$

Feasible                      Unsure                      Infeasible
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,

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

<table>
<thead>
<tr>
<th>( n )</th>
<th>( UB(RM, n) )</th>
<th>( n )</th>
<th>( UB(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>( \rightarrow \infty )</td>
<td>( \ln 2 \geq 0.693 )</td>
</tr>
</tbody>
</table>
Utilization Bound in Terms of $n$
Proof Sketch for $UB(RM, n)$

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://www.ulb.ac.be/di/ssd/goossens/lub.ps. Note the proof we present is a bit different than the one presented by Buttazzo’s textbook. Without loss of generality (why?), we will only consider task sets with distinct periods, i.e., $T_1 < T_2 < \cdots < T_n$. We will present our proof sketch in two parts:

1. First, we consider the special case where $T_n \leq 2T_1$.
2. Second, we show how to relax this constraint.
Proof Sketch: Difficult-To-Schedule

Definition

A task set $\mathcal{T}_n$ is called **difficult-to-schedule** under scheduling algorithm $A$ if $\mathcal{T}_n$ **fully utilizes** the processor if scheduled under $A$.

Our Strategy

- We seek the most **difficult-to-schedule** task set $\mathcal{T}_n$ for $n$ tasks: A task set that is **difficult-to-schedule** with the **minimal utilization**.
- We derive a tight lower bound on the utilization of the most **difficult-to-schedule** task set $\mathcal{T}_n$ in terms of $n$, the $UB(RM, n)$. 
Proof Sketch: Three Steps

1. For a task set with given periods, define the execution times for the most difficult-to-schedule task set.
2. Show that any difficult-to-schedule task set whose execution times differ from those in step 1 has a greater utilization than the most difficult-to-schedule task set provided in step 1.
3. Compute a closed-form expression for $UB(RM, n)$. 
Step 1: Execution Time

Suppose $T_n^*$ is a task set with $T_n \leq 2T_1$ and

$$C_k = T_{k+1} - T_k \quad \text{for } k = 1, 2, \ldots, n - 1$$

$$C_n = T_n - 2 \sum_{k=1}^{n-1} C_k = 2T_1 - T_n. \quad \text{(Because: } \sum_{k=1}^{n-1} C_k = T_n - T_1)$$

Theorem

Such a task set $T_n^*$ is the most difficult-to-schedule task set.
Proof of the Most Difficult-to-Schedule Statement

Proof strategy:
We show that the difficult-to-schedule task set given on the previous slide can be transformed into any difficult-to-schedule task set without decreasing the task set’s utilization:

Starting with the highest priority task and working our way down to the lowest priority task, we incrementally modify the execution times of $T_n^*$ to match any other difficult-to-schedule task set, and for each modification, utilization does not decrease.

If we need to increase (decrease) the execution time of task $\tau_i$, then we compensate for this by decreasing (increasing) the execution time of some $\tau_k$, where $k > i$. 
Proof of the Most Difficult-to-Schedule:
Increase by \( \Delta \)

Let’s increase the execution of some task \( \tau_i \) (\( i < n \)) by \( \Delta \) (\( \Delta > 0 \)).

\[
C'_i = T_{i+1} - T_i + \Delta = C_i + \Delta.
\]

To keep the processor busy up to \( T_n \), we can decrease the execution time of a task \( \tau_k \) (\( k > i \)) by \( \Delta \).

\[
C'_k = C_k - \Delta.
\]

Since \( T_i < T_k \), the utilization of the above task set \( \mathcal{T}'_n \) is no less than the original task set \( \mathcal{T}^*_n \) by

\[
U(\mathcal{T}'_n) - U(\mathcal{T}^*_n) = \left( \frac{\Delta}{T_i} \right) - \frac{\Delta}{T_k} \geq 0.
\]
Proof of the Most Difficult-to-Schedule: Decrease by \( \Delta \)

Let’s decrease the execution of some task \( \tau_i \) \( (i < n) \) by \( \Delta \) \( (\Delta > 0) \).

\[
C_i' = T_{i+1} - T_i - \Delta = C_i - \Delta.
\]

To keep the processor busy up to \( T_n \), we can increase the execution time of a task \( \tau_k \) \( (k > i) \) by \( 2\Delta \). (Why \( 2\Delta \)?)

\[
C_k' = C_k + 2\Delta.
\]

Since \( T_k \leq 2T_i \), the utilization of the above task set \( \mathcal{T}_n' \) is no less than the original task set \( \mathcal{T}_n^* \) by

\[
U(\mathcal{T}_n') - U(\mathcal{T}_n^*) = \left[ -\frac{\Delta}{T_i} + \frac{2\Delta}{T_k} \right] \geq 0.
\]

\[
\Rightarrow -\frac{\Delta}{T_i} + \frac{2\Delta}{2T_i} = 0
\]
Calculating $UB(RM, n) =$

The Utilization of the Most Difficult-to-Schedule Task Set

- Utilization of $T^n_*$ for given task periods:
  Let $x_i$ be $\frac{T_{i+1}}{T_i}$.
  
  $$U(T^n_*) = \frac{2T_1 - T_n}{T_n} + \sum_{i=1}^{n-1} \frac{T_{i+1} - T_i}{T_i} = 2 \left(\frac{T_1}{T_n}\right) + \left(\sum_{i=1}^{n-1} x_i\right) - n$$
  
  $$= \frac{2}{\prod_{i=1}^{n-1} x_i} + \left(\sum_{i=1}^{n-1} x_i\right) - n$$

- Which task periods minimize the utilization?
  By getting partial derivatives of $U(T^n_*)$ to the variables, we know that $U(T^n_*)$ is minimized when
  
  $$\frac{\partial U(T^n_*)}{\partial x_k} = 1 - 2 \left(\frac{\prod_{i=1}^{n-1} x_i/x_k}{(\prod_{i=1}^{n-1} x_i)^2}\right) = 1 - \frac{2}{x_k \prod_{i=1}^{n-1} x_i} = 0, \forall k = 1, 2, \ldots, n-1.$$ 

  Therefore, all $x_i$ need to be equal for $k = 1, 2, \ldots, n - 1$: $x_1 = x_2 = \cdots = x_{n-1}$:

  $$x_k = \frac{2}{\prod_{i=1}^{n-1} x_i} \Rightarrow x_k^n = 2 \Rightarrow x_k = 2^{\frac{1}{n}}.$$
Calculating $UB(RM, n)$ (cont.)

- By substituting $x_i = 2^{\frac{1}{n}}$ into our utilization formula, we have

\[ U(T_n^*) \leq n(2^{\frac{1}{n}} - 1) \]
Removing the Constraint on $T_n \leq 2T_1$

**Strategy**

- Suppose that $T_n$ is a difficult-to-schedule task set with $n$ tasks.
- We are going to transform any difficult-to-schedule task set $T_n$ with $n$ tasks and $T_n > 2T_i$ for some $i$ to $T'_n$ such that
  1. the period $T_n$ in $T'_n$ is no more than twice of $T_1$ in $T'_n$, and
  2. $U(T_n) \geq U(T'_n)$.

Transform $T_n$ step-by-step to get $T'_n$:

- Find a task $\tau_k$ in task set $T_n$ with $\ell T_k < T_n \leq (\ell + 1) T_k$ and $\ell$ is an integer that is at least 2.
- Create a task $\tau'_n$ such that the period is $T_n$ and the execution time $C'_n$ is $C_n + (\ell - 1)C_k$.
- Create a task $\tau'_k$ such that the period is $\ell T_k$ and the execution time $C'_k$ is $C_k$.
- Let $T'_n$ be $T_n \setminus \{\tau_k, \tau_n\} \cup \{\tau'_k, \tau'_n\}$.
Removing the Constraint on $T_n \leq 2T_1$ (cont.)

Conditions:

1. $\ell T_k < T_n \leq (\ell + 1) T_k$ with $\ell \geq 2$.
2. $T'_n = T_n$, $C'_n = C_n + (\ell - 1)C_k$ and $T'_k = \ell T_k$ and $C'_k = C_k$.

Results: since $\ell T_k < T_n$, we know

$$U(T_n) - U(T'_n) = \frac{C_n}{T_n} + \frac{C_k}{T_k} - \frac{(\ell - 1)C_k + C_n}{T_n} - \frac{C_k}{\ell T_k}$$

$$= \left( \frac{1}{\ell T_k} - \frac{1}{T_n} \right) (\ell - 1)C_k > 0$$
Concluding the RM Analysis

- Moreover \( T'_n \) above is also a difficult-to-schedule task set.
- By repeating the above procedure, we can transform to a task set \( T'_n \) with \( T'_n \leq 2T'_1 \) without increasing its utilization.

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,

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

\[
\lim_{n \to \infty} UB(RM, n) = \lim_{n \to \infty} n(2\frac{1}{n} - 1) = \ln 2 \geq 0.693
\]
Remarks on Harmonic Task Set

- If the total utilization is larger than 0.693 but less than or equal to 1, the utilization-bound schedulability test 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.
Improving Schedulability: Hyperbolic Bound

Theorem

[Bini and Buttazzo, ECRTS 2001] A system 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

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

Note that this is also only a sufficient schedulability test.
Recall Most Difficult-to-Schedule Task Set

Suppose $T^*_n$ is a task set with $T_n \leq 2T_1$ and

$$C_k = T_{k+1} - T_k$$

for $k = 1, 2, \ldots, n - 1$

$$C_n = T_n - 2 \sum_{k=1}^{n-1} C_k = 2T_1 - T_n.$$  

Let $x_i$ be $\frac{T_{i+1}}{T_i}$.

- For $i = 1, 2, \ldots, n - 1$, we have $U_i = x_i - 1 \Rightarrow U_i + 1 = x_i$.
- $U_n = 2 \frac{T_1}{T_n} - 1 \Rightarrow U_n + 1 = 2 \frac{T_1}{T_n}$.

$$\prod_{i=1}^n(U_i + 1) = \left(\frac{T_2}{T_1} \frac{T_3}{T_2} \cdots \frac{T_n}{T_{n-1}}\right) \cdot 2 \frac{T_1}{T_n}$$

$$= 2.$$
Hyperbolic Bound vs Utilization Bound
Outline

1 Schedulability Analysis for Static-Priority Scheduling
   - Utilization-Based Analysis (Relative Deadline = Period)
   - Demand-Based Analysis

2 Schedulability Analysis for Dynamic-Priority Scheduling
Necessary and Sufficient RM-Schedulability

- Time-demand analysis (TDA) was proposed by Lehoczky, Sha, and Ding.
- 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 is necessary and sufficient.
Schedulability Condition

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, preemttable tasks is schedulable on one processor by algorithm A if it holds that:

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

This condition is also necessary for synchronous, periodic task sets and also sporadic 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 times $t$ 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 $t(0) := \sum_{j=1}^{i} C_j$, and stopping when, for some $\ell$, $t(\ell) = W_i(t(\ell))$ or $t(\ell) > D_i$.

- Only consider $t \in \{\ell T_j - \epsilon\} 1 \leq j \leq i, \ell \in \mathcal{N}^+$, where $\epsilon$ is a constant close to 0. That is, only consider $t$ at which a job of higher-priority tasks arrives.
Complexity of TDA Analysis

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 pseudo-polynomial time complexity.

Theorem

Outline

1. Schedulability Analysis for Static-Priority Scheduling
   - Utilization-Based Analysis (Relative Deadline = Period)
   - Demand-Based Analysis

2. Schedulability Analysis 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.

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

$$d_{i,k} - t_{-1} < \text{demand in } [t_{-1}, d_{i,k}) \text{ by jobs with deadline no more than } d_{i,k}$$

$$= \sum_{a_k \geq t_{-1}, d_k \leq d_{i,k}} C_k$$

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

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

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

$$d_{i,k} - t_{-1} < \text{demand in } [t_{-1}, d_{i,k}) \text{ by jobs with deadline no more than } d_{i,k}$$

$$= \sum_{a_k \geq t_{-1}, d_k \leq d_{i,k}} C_k$$

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

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

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

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

Theorem

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

$$\sum_{k=1}^{n} \frac{C_k}{D_k} \leq 1.$$

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

**Theorem**

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

\[
dbf(\tau_i, t) = \max \left\{ 0, \left\lceil \frac{t + T_i - D_i}{T_i} \right\rceil \right\} \cdot C_i = \max \left\{ 0, \left\lceil \frac{t - D_i}{T_i} \right\rceil + 1 \right\} \cdot C_i.
\]

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 and only if \( \forall L \geq 0, \sum_{i=1}^{n} dbf(\tau_i, L) \leq L. \)
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

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

\# of jobs with deadline no more than \(t_2\)

\# of jobs with arrival time less than \(t_1\)

- 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.,

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

- 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 $LCM(T_1, T_2, \ldots, T_n)$.

- It takes pseudo-polynomial time.

**Theorem**

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

**RM**
- Low run-time overhead: $O(1)$ with priority sorting in advance
- Optimal for static priority
- Schedulability test is $NP$-hard (even if the relative deadline = period)
- Utilization bound: 0.693
- In general, more preemptions

**EDF**
- High run-time overhead: $O(\log n)$ with balanced binary tree
- Optimal for dynamic priority
- Schedulability test is easy (when the relative deadline = period)
- Utilization bound: 1
- In general, fewer preemptions