EDF Scheduling

Giuseppe Lipari
http://feanor.sssup.it/~lipari

Scuola Superiore Sant'Anna – Pisa

May 11, 2008
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Earliest Deadline First

- An important class of scheduling algorithms is the class of *dynamic priority* algorithms
  - In dynamic priority algorithms, the priority of a task can change during its execution
  - Fixed priority algorithms are a sub-class of the more general class of dynamic priority algorithms: the priority of a task does not change.

- The most important (and analyzed) dynamic priority algorithm is Earliest Deadline First (EDF)
  - The priority of a job (instance) is inversely proportional to its absolute deadline;
  - In other words, the highest priority job is the one with the earliest deadline;
  - If two tasks have the same absolute deadlines, chose one of the two at random (*ties can be broken arbitrarily*).
  - The priority is dynamic since it changes for different jobs of the same task.
Example: scheduling with RM

- We schedule the following task set with FP (RM priority assignment).
- $\tau_1 = (1, 4), \tau_2 = (2, 6), \tau_4 = (3, 8)$.
- $U = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{23}{24}$
- The utilization is greater than the bound: there is a deadline miss!

Observe that at time 6, even if the deadline of task $\tau_3$ is very close, the scheduler decides to schedule task $\tau_2$. This is the main reason why $\tau_3$ misses its deadline!
Example: scheduling with EDF

- Now we schedule the same task set with EDF.
  - $\tau_1 = (1, 4), \tau_2 = (2, 6), \tau_4 = (3, 8)$.
  - $U = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{23}{24}$
  - Again, the utilization is very high. However, no deadline miss in the hyperperiod.

Observe that at time 6, the problem does not appear, as the earliest deadline job (the one of $\tau_3$) is executed.
Job-level fixed priority

- In EDF, the priority of a job is *fixed*.
- Therefore some author is classifies EDF as of *job-level fixed priority* scheduling;
- LLF is a *job-level dynamic priority* scheduling algorithm as the priority of a job may vary with time;
- Another job-level dynamic priority scheduler is p-fair.
Outline

1. Dynamic priority

2. Basic analysis

3. FP vs EDF

4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm

5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
A general approach to schedulability analysis

We start from a completely aperiodic model.

- A system consists of a (infinite) set of jobs
  \[ J = \{ J_1, J_2, \ldots, J_n, \ldots \} \].
- \( J_k = (a_k, c_k, d_k) \)
- Periodic or sporadic task sets are particular cases of this system
EDF optimality

Theorem (Dertouzos ’73)
If a set of jobs $\mathcal{J}$ is schedulable by an algorithm A, then it is schedulable by EDF.

Proof.
The proof uses the exchange method.
- Transform the schedule $\sigma_A(t)$ into $\sigma_{EDF}(t)$, step by step;
- At each step, preserve schedulability.

Corollary
EDF is an optimal algorithm for single processors.
Schedulability bound for periodic/sporadic tasks

Theorem

Given a task set of periodic or sporadic tasks, with relative deadlines equal to periods, the task set is schedulable by EDF if and only if

\[ U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \]

Corollary

EDF is an optimal algorithm, in the sense that if a task set is schedulable, then it is schedulable by EDF.

Proof.

In fact, if \( U > 1 \) no algorithm can successfully schedule the task set; if \( U \leq 1 \), then the task set is schedulable by EDF (and maybe by other algorithms).
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Advantages of EDF over FP

- EDF can schedule all task sets that can be scheduled by FP, but not vice versa.
  - Notice also that offsets are not relevant!
- There is not need to define priorities
  - Remember that in FP, in case of offsets, there is not an optimal priority assignment that is valid for all task sets
- In general, EDF has less context switches
  - In the previous example, you can try to count the number of context switches in the first interval of time: in particular, at time 4 there is no context switch in EDF, while there is one in FP.
- Optimality of EDF
  - We can fully utilize the processor, less idle times.
Disadvantages of EDF over FP

- EDF is not provided by any commercial RTOS, because of some disadvantage
- Less predictable
  - Looking back at the example, let’s compare the response time of task $\tau_1$: in FP is always constant and minimum; in EDF is variable.
- Less controllable
  - if we want to reduce the response time of a task, in FP is only sufficient to give him an higher priority; in EDF we cannot do anything;
  - We have less control over the execution
More implementation overhead

- FP can be implemented with a very low overhead even on very small hardware platforms (for example, by using only interrupts);
- EDF instead requires more overhead to be implemented (we have to keep track of the absolute deadline in a long data structure);
- There are methods to implement the queueing operations in FP in $O(1)$; in EDF, the queueing operations take $O(\log N)$, where $N$ is the number of tasks.
Domino effect

- In case of overhead \((U > 1)\), we can have the *domino effect* with EDF: it means that all tasks miss their deadlines.

- An example of domino effect is the following:

  ![Graph showing domino effect]

  - All tasks missed their deadline almost at the same time.
Domino effect: considerations

- FP is more predictable: only lower priority tasks miss their deadlines! In the previous example, if we use FP:

- As you can see, while $\tau_1$ and $\tau_2$ never miss their deadlines, $\tau_3$ misses a lot of deadline, and $\tau_4$ does not execute!

- However, it may happen that some task never executes in case of high overload, while EDF is more fair (all tasks are treated in the same way).
Response time computation

Computing the response time in EDF is very difficult, and we will not present it in this course.

- In FP, the response time of a task depends only on its computation time and on the interference of higher priority tasks.
- In EDF, it depends in the parameters of all tasks!
- If all offset are 0, in FP the maximum response time is found in the first job of a task,
- In EDF, the maximum response time is not found in the first job, but in a later job.
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Generalization to deadlines different from period

- EDF is still optimal when relative deadlines are not equal to the periods.
- However, the schedulability analysis formula becomes more complex.
- If all relative deadlines are less than or equal to the periods, a first trivial (sufficient) test consist in substituting $T_i$ with $D_i$:

$$U' = \sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1$$

- In fact, if we consider each task as a sporadic task with interarrival time $D_i$ instead of $T_i$, we are increasing the utilization, $U < U'$. If it is still less than 1, then the task set is schedulable. If it is larger than 1, then the task set may or may not be schedulable.
In the following slides, we present a general methodology for schedulability analysis of EDF scheduling.

Let’s start from the concept of **demand function**

**Definition:** the demand function for a task $\tau_i$ is a function of an interval $[t_1, t_2]$ that gives the amount of computation time that *must* be completed in $[t_1, t_2]$ for $\tau_i$ to be schedulable:

$$df_i(t_1, t_2) = \sum_{a_{ij} \geq t_1 \land d_{ij} \leq t_2} c_{ij}$$

For the entire task set:

$$df(t_1, t_2) = \sum_{i=0}^{N} df_i(t_1, t_2)$$
Example of demand function

- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$

Let’s compute $df()$ in some intervals;
Example of demand function

\[ \tau_1 = (1, 4, 6), \ \tau_2 = (2, 6, 8), \ \tau_3 = (3, 5, 10) \]

Let’s compute \( df() \) in some intervals;

\[ df(7, 22) = 2 \cdot C_1 + 2 \cdot C_2 + 1 \cdot C_3 = 9; \]
Example of demand function

\( \tau_1 = (1, 4, 6), \ \tau_2 = (2, 6, 8), \ \tau_3 = (3, 5, 10) \)

Let’s compute \( df() \) in some intervals;
- \( df(7, 22) = 2 \cdot C_1 + 2 \cdot C_2 + 1 \cdot C_3 = 9 \);
- \( df(3, 13) = 1 \cdot C_1 = 1 \);
Example of demand function

\[ \tau_1 = (1, 4, 6), \quad \tau_2 = (2, 6, 8), \quad \tau_3 = (3, 5, 10) \]

Let's compute \( df() \) in some intervals:

\[ df(7, 22) = 2 \cdot C_1 + 2 \cdot C_2 + 1 \cdot C_3 = 9; \]
\[ df(3, 13) = 1 \cdot C_1 = 1; \]
\[ df(10, 25) = 2 \cdot C_1 + 1 \cdot C_2 + 2 \cdot C_3 = 7; \]
A necessary condition

**Theorem**

A necessary condition for any job set to be schedulable by any scheduling algorithm when executed on a single processor is that:

\[ \forall t_1, t_2 \quad df(t_1, t_2) \leq t_2 - t_1 \]

**Proof.**

By contradiction. Suppose that \( \exists t_1, t_2 \quad df(t_1, t_2) > t_2 - t_1 \). If the system is schedulable, then it exists a scheduling algorithm that can execute more than \( t_2 - t_1 \) units of computations in an interval of length \( t_2 - t_1 \). Absurd!
Main theorem

Theorem

A necessary and sufficient condition for a set of jobs $J$ to be schedulable by EDF is that

$$\forall t_1, t_2 \quad df(t_1, t_2) \leq t_2 - t_1$$

(1)

Proof.

The proof is based on the same technique used by Liu & Layland in their seminal paper. We only need to prove the sufficient part.
Main theorem

**Theorem**

*A necessary and sufficient condition for a set of jobs $\mathcal{J}$ to be schedulable by EDF is that*

$$\forall t_1, t_2 \quad df(t_1, t_2) \leq t_2 - t_1 \quad (1)$$

**Proof.**

The proof is based on the same technique used by Liu & Layland in their seminal paper. We only need to prove the *sufficient* part.

- By contradiction: assume a deadline is missed and the condition holds.
Main theorem

Theorem

A necessary and sufficient condition for a set of jobs $J$ to be schedulable by EDF is that

$$\forall t_1, t_2 \quad df(t_1, t_2) \leq t_2 - t_1$$

(1)

Proof.

The proof is based on the same technique used by Liu & Layland in their seminal paper. We only need to prove the sufficient part.

- By contradiction: assume a deadline is missed and the condition holds
- Assume the first deadline miss is at $y$
Main theorem

Theorem

A necessary and sufficient condition for a set of jobs \( J \) to be schedulable by EDF is that

\[ \forall t_1, t_2 \quad df(t_1, t_2) \leq t_2 - t_1 \quad (1) \]

Proof.

The proof is based on the same technique used by Liu & Layland in their seminal paper. We only need to prove the sufficient part.

- By contradiction: assume a deadline is missed and the condition holds
- Assume the first deadline miss is at \( y \)
- We find an opportune \( x < y \) such that \( df(x, y) > y - x \).
Proof

Suppose the first deadline miss is at time $y$. Let $x$ be the **last instant prior to** $y$ such that:

- all jobs with arrival time before $x$ and deadline before $y$ have already completed by $x$;
- $x$ coincides with the arrival time of a job with deadline less or equal to $y$
- Such instant always exists (it could be time 0).
Proof

Suppose the first deadline miss is at time $y$. Let $x$ be the last instant prior to $y$ such that:

- all jobs with arrival time before $x$ and deadline before $y$ have already completed by $x$;
- $x$ coincides with the arrival time of a job with deadline less or equal to $y$
- Such instant always exists (it could be time 0).

Since $x$ is the last such instant, it follows that:
- there is no idle time in $[x, y]$
Suppose the first deadline miss is at time $y$. Let $x$ be the last instant prior to $y$ such that:

- all jobs with arrival time before $x$ and deadline before $y$ have already completed by $x$;
- $x$ coincides with the arrival time of a job with deadline less of equal to $y$.
Such instant always exists (it could be time 0).

Since $x$ is the last such instant, it follows that:

- there is no idle time in $[x, y]$
Proof

- Suppose the first deadline miss is at time \( y \). Let \( x \) be the **last instant prior to** \( y \) such that:
  - all jobs with arrival time before \( x \) and deadline before \( y \) have already completed by \( x \);
  - \( x \) coincides with the arrival time of a job with deadline less of equal to \( y \)
  - Such instant always exists (it could be time 0).
- Since \( x \) is the last such instant, it follows that:
  - there is no idle time in \([x, y]\)
  - No job with deadline greater than \( y \) executes in \([x, y]\)
  - only jobs with arrival time greater or equal to \( x \), and deadline less than or equal to \( y \) execute in \([x, y]\)
Proof

- Suppose the first deadline miss is at time $y$. Let $x$ be the **last instant prior to** $y$ such that:
  - all jobs with arrival time before $x$ and deadline before $y$ have already completed by $x$;
  - $x$ coincides with the arrival time of a job with deadline less of equal to $y$
  - Such instant always exists (it could be time 0).

- Since $x$ is the last such instant, it follows that:
  - there is no idle time in $[x, y]$
  - No job with deadline greater than $y$ executes in $[x, y]$
  - only jobs with arrival time greater or equal to $x$, and deadline less than or equal to $y$ execute in $[x, y]$

- Since there is a deadline miss in $[x, y]$, $df(x, y) > y - x$, and the theorem follows.
Feasibility analysis

- The previous theorem gives a first hint at how to perform a schedulability analysis.
  
  - However, the condition should be checked for all pairs $[t_1, t_2]$.
  - This is impossible in practice! (an infinite number of intervals!).
  - First observation: function df changes values only at discrete instants, corresponding to arrival times and deadline of a job set.
Feasibility analysis

- The previous theorem gives a first hint at how to perform a schedulability analysis.
  - However, the condition should be checked for all pairs \([t_1, t_2]\).
  - This is impossible in practice! (an infinite number of intervals!).
  - First observation: function \(df\) changes values only at discrete instants, corresponding to arrival times and deadline of a job set.
  - Second, for periodic tasks we could use some periodicity (hyperperiod) to limit the number of points to be checked to a finite set.
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Simplifying the analysis

- A periodic task set is *synchronous* if all task offsets are equal to 0.
- In other words, for a synchronous task set, all tasks start at time 0.
- A task set is *asynchronous* if some task has a non-zero offset.
Theorem

For a set of synchronous periodic tasks (i.e. with no offset),

\[ \forall t_1, t_2 > t_1 \quad df(t_1, t_2) \leq df(0, t_2 - t_1) \]

- In plain words, the worst case demand is found for intervals starting at 0.
- **Definition:** Demand Bound function:

\[ dbf(L) = \max_t (df(t, t + L)) = df(0, L). \]
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$dbf_i(L) = \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right)_0 C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$\text{dbf}_i(L) = \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right)_{0} C_i$$
Demand bound function - II

- The maximum is when the task is activated at the beginning of the interval.
- For a periodic task $\tau_i$:

$$dbf_i(L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right)_0 C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$dbf_i(L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$dbf_i(L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right)_{0} C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$: 

$$dbf_i(L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$\text{dbf}_i(L) = \left( \left[ \frac{L - D_i}{T_i} \right] + 1 \right)_0 C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$\text{dbf}_i(L) = \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$
Demand bound function - II

- The maximum is when the task is activated at the beginning of the interval.
- For a periodic task $\tau_i$:

$$dbf_i(L) = \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right)_0 C_i$$
Demand bound function - II

- The maximum is when the task is activated at the beginning of the interval.
- For a periodic task $\tau_i$:

$$dbf_i(L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$
The maximum is when the task is activated at the beginning of the interval.

For a periodic task $\tau_i$:

$$\text{dbf}_i(L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right)_0 C_i$$
Theorem (Baruah, Howell, Rosier ’90)

A synchronous periodic task set \( T \) is schedulable by EDF if and only if:

\[
\forall L \in \text{dead}(T) \quad \text{dbf}(L) \leq L
\]

where \( \text{dead}(T) \) is the set of deadlines in \([0, H]\)

Proof next slide.
Proof

- Sufficiency: eq. holds $\rightarrow$ task set is schedulable.
  - By contradiction

- Necessity: task set is schedulable $\rightarrow$ eq. holds
Proof

- **Sufficiency**: eq. holds $\rightarrow$ task set is schedulable.
  - By contradiction
    - If deadline is missed in $y$, then $\exists x, y \ y - x < df(x, y)$

- **Necessity**: task set is schedulable $\rightarrow$ eq. holds
Proof

- **Sufficiency**: eq. holds $\rightarrow$ task set is schedulable.
  - By contradiction
    - If deadline is missed in $y$, then $\exists x, y \quad y - x < df(x, y)$
    - it follows that $y - x < df(x, y) \leq dbf(y - x)$
  \[\square\]

- **Necessity**: task set is schedulable $\rightarrow$ eq. holds
Proof

- **Sufficiency**: eq. holds $\rightarrow$ task set is schedulable.
  - By contradiction
    - If deadline is missed in $y$, then $\exists x, y \ y - x < df(x, y)$
    - it follows that $y - x < df(x, y) \leq dbf(y - x)$

- **Necessity**: task set is schedulable $\rightarrow$ eq. holds
  - By contradiction
Proof

- **Sufficiency:** eq. holds $\rightarrow$ task set is schedulable.
  - By contradiction
    - If deadline is missed in $y$, then $\exists x, y \ y - x < df(x, y)$
    - it follows that $y - x < df(x, y) \leq dbf(y - x)$

- **Necessity:** task set is schedulable $\rightarrow$ eq. holds
  - By contradiction
    - eq. does not hold for $\overline{L}$. 

□
Proof

• Sufficiency: eq. holds $\rightarrow$ task set is schedulable.
  • By contradiction
  • If deadline is missed in $y$, then $\exists x, y \ y - x < df(x, y)$
  • it follows that $y - x < df(x, y) \leq dbf(y - x)$ □

• Necessity: task set is schedulable $\rightarrow$ eq. holds
  • By contradiction
  • eq. does not hold for $\bar{L}$.
  • build a schedule starting at 0, for which $dbf(\bar{L}) = df(0, \bar{L})$
Proof

- **Sufficiency:** eq. holds $\rightarrow$ task set is schedulable.
  - By contradiction
    - If deadline is missed in $y$, then $\exists x, y \ y - x < df(x, y)$
    - it follows that $y - x < df(x, y) \leq dbf(y - x)$

- **Necessity:** task set is schedulable $\rightarrow$ eq. holds
  - By contradiction
    - eq. does not hold for $L$.
    - build a schedule starting at 0, for which $dbf(\bar{L}) = df(0, \bar{L})$
    - Hence task set is not schedulable
Sporadic task

- Sporadic tasks are equivalent to synchronous periodic task sets.
- For them, the worst case is when they all arrive at their maximum frequency and starting synchronously.
Synchronous and asynchronous

Let $\mathcal{T}$ be an asynchronous task set.
We call $\mathcal{T}'$ the corresponding synchronous set, obtained by setting all offset equal to 0.

Corollary

If $\mathcal{T}'$ is schedulable, then $\mathcal{T}$ is schedulable too.

Conversely, if $\mathcal{T}$ is schedulable, $\mathcal{T}'$ may not be schedulable.

The proof follows from the definition of $\text{dbf}(L)$.
A pseudo-polynomial test

Theorem (Baruah, Howell, Rosier, ’90)

Given a synchronous periodic task set $T$, with deadlines less than or equal to the period, and with load $U < 1$, the system is schedulable by EDF if and only if:

$$\forall L \in \text{deadShort}(T) \quad \text{dbf}(L) \leq L$$

where $\text{deadShort}(T)$ is the set of all deadlines in interval $[0, L^*]$ and

$$L^* = \frac{U}{1-U} \max_i (T_i - D_i)$$

Corollary

The complexity of the above analysis is pseudo-polynomial.
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Example of computation of the $dbf$

- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = \frac{1}{6} + \frac{1}{4} + \frac{3}{10} = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.

Let’s compute $dbf()$
Example of computation of the *dbf*

- $\tau_1 = (1, 4, 6), \tau_2 = (2, 6, 8), \tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167, L^* = 12.64.$
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.

Let's compute $dbf()$
- $df(0, 4) = C_1 = 1 < 4$;
Example of computation of the $dbf$

- $\tau_1 = (1, 4, 6), \tau_2 = (2, 6, 8), \tau_3 = (3, 5, 10)$
- $U = \frac{1}{6} + \frac{1}{4} + \frac{3}{10} = 0.7167, L^* = 12.64.$
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.

Let’s compute $dbf()$

- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;
Example of computation of the $dbf$

- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.

Let’s compute $dbf()$

- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;
- $df(0, 6) = C_1 + C_2 + C_3 = 6 \leq 6$;
Example of computation of the \textit{dbf}

- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.

Let’s compute $\textit{dbf}()$

- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;
- $df(0, 6) = C_1 + C_2 + C_3 = 6 \leq 6$;
- $df(0, 10) = 2C_1 + C_2 + C_3 = 7 \leq 10$;
Example of computation of the dbf

- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.

Let’s compute $dbf()$

- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;
- $df(0, 6) = C_1 + C_2 + C_3 = 6 \leq 6$;
- $df(0, 10) = 2C_1 + C_2 + C_3 = 7 \leq 10$;
- The task set is schedulable.
Idle time and busy period

- The interval between time 0 and the first idle time is called *busy period*.
- The analysis can be stopped at the first idle time (Spuri, ’94).
- The first idle time can be found with the following recursive equations:

\[
W(0) = \sum_{i=1}^{N} C_i
\]

\[
W(k) = \sum_{i=1}^{N} \left\lceil \frac{W(k-1)}{T_i} \right\rceil C_i
\]

- The iteration stops when \( W(k - 1) = W(k) \).
Another example

Consider the following example

<table>
<thead>
<tr>
<th>τ</th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4.5</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>

$U = 0.9; L^* = 9 \times 7 = 63;$
Another example

Consider the following example

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4.5</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>

$U = 0.9; \ L^* = 9 \times 7 = 63;$

$W = 14.5.$
Another example

Consider the following example

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4.5</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>

$U = 0.9$; $L^* = 9 \times 7 = 63$;

$W = 14.5$.

Then we can check all deadline in interval $[0, 14.5]$. 
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Algorithm

- Of course, it should not be necessary to draw the schedule to see if the system is schedulable or not.
- First of all, we need a formula for the $dbf$:

$$dbf(L) = \sum_{i=1}^{N} \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

- The algorithm works as follows:
  - We list all deadlines of all tasks until $L^*$.  
  - Then, we compute the $dbf$ for each deadline and verify the condition.
The previous example

- In the previous example: deadlines of the tasks:

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>4</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

- dbf in tabular form

<table>
<thead>
<tr>
<th>L</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbf</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

- Since, for all $L < L^*$ we have $dbf(L) \leq L$, then the task set is schedulable.
Another example

Consider the following task set

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4.5</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>

$U = 0.9; \quad L^* = 9 \times 7 = 63$;

hint: if $L^*$ is too large, we can stop at the first idle time.

The first idle time can be found with the following recursive equations:

$$W(0) = \sum_{i=1}^{N} C_i$$

$$W(k) = \sum_{i=1}^{N} \left[ \frac{W(k-1)}{T_i} \right] C_i$$

The iteration stops when $W(k-1) = W(k)$.

In our example $W = 14.5$. Then we can check all deadlines in interval $[0, 14.5]$. 
Example

- Deadlines of the tasks:

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>2</th>
<th>6</th>
<th>10</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>9</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Demand bound function in tabular form

<table>
<thead>
<tr>
<th>t</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dbf$</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>8.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The task set is not schedulable! Deadline miss at 8.
In the schedule...

The schedule is as follows:
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Outline

1. Dynamic priority
2. Basic analysis
3. FP vs EDF
4. Processor demand bound analysis
   - Generalization to deadlines different from period
   - Synchronous and asynchronous tasks
   - Examples
   - Testing algorithm
5. A sufficient pseudo-polynomial test for synchronous sets
   - Basic idea
Differences between synchronous and asynchronous sets

Let’s recall the previous Corollary and Theorem
Differences between synchronous and asynchronous sets

- Let’s recall the previous Corollary and Theorem
- Let us analyze the reasons why.
- When computing $\text{dbf}(L)$ we do the following steps:
  - Consider any interval $[t_1, t_2]$ of length $L$
  - ”push back” activations until the first jobs starts at $t_1$;
  - Compute the dbf as the sum of the computation of all jobs with deadline no later than $t_2$. 
Differences between synchronous and asynchronous sets

- Let’s recall the previous Corollary and Theorem
- Let us analyze the reasons why.
- When computing $\text{dbf}(L)$ we do the following steps:
  - Consider any interval $[t_1, t_2]$ of length $L$
  - "push back" activations until the first job starts at $t_1$
  - Compute the dbf as the sum of the computation of all jobs with deadline no later than $t_2$. 
Differences between synchronous and asynchronous sets

- Let’s recall the previous Corollary and Theorem
- Let us analyze the reasons why.
- When computing dbf(\(L\)) we do the following steps:
  - Consider any interval \([t_1, t_2]\) of length \(L\)
  - "push back" activations until the first jobs starts at \(t_1\);
  - Compute the dbf as the sum of the computation of all jobs with deadline no later than \(t_2\).

**Problem:** by “pushing back” the instance we are modifying the task set!
Example of asynchronous task set

\[ \tau_1 = (0, 4, 7, 9) \text{ and } \tau_2 = (2, 5, 8, 12) \]

\[ df(0, 8) = 4 \]
Example of asynchronous task set

\( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)

- \( \text{df}(0, 8) = 4 \)
- \( \text{df}(2, 10) = 5 \)
Example of asynchronous task set

- $\tau_1 = (0, 4, 7, 9)$ and $\tau_2 = (2, 5, 8, 12)$
- $df(0, 8) = 4$
- $df(2, 10) = 5$
Example of asynchronous task set

\( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)
Example of asynchronous task set

\[ \tau_1 = (0, 4, 7, 9) \text{ and } \tau_2 = (2, 5, 8, 12) \]
Example of asynchronous task set

\[ \tau_1 = (0, 4, 7, 9) \text{ and } \tau_2 = (2, 5, 8, 12) \]
Example of asynchronous task set

\[ \tau_1 = (0, 4, 7, 9) \text{ and } \tau_2 = (2, 5, 8, 12) \]
Example of asynchronous task set

\( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)
Example of asynchronous task set

\( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)

\( \text{dbf}(8) = 9 \)
Example of asynchronous task set

\[ \tau_1 = (0, 4, 7, 9) \text{ and } \tau_2 = (2, 5, 8, 12) \]

\[ \text{dbf}(8) = 9 \]

The dbf is too pessimistic.
Trade off between pessimism and complexity

- The problem is that we do not know what is the worst pattern of arrivals for asynchronous task sets.
- We know for synchronous: instant 0
- For asynchronous, we should check for every possible pattern
Key observation

- The distance between any arrival of task $\tau_i$ and any arrival of task $\tau_j$ is:

$$a_{j,k_1} - a_{i,k_2} = \phi_j + k_1 T_j - \phi_i - k_2 T_i = \phi_j - \phi_i + k(\gcd(T_i, T_j))$$
Key observation

- The **distance** between any arrival of task $\tau_i$ and any arrival of task $\tau_j$ is:

$$a_{j,k_1} - a_{i,k_2} = \phi_j + k_1 T_j - \phi_i - k_2 T_i = \phi_j - \phi_i + k(\gcd(T_i, T_j))$$

- Imposing that the difference must not be negative, and $k$ must be integer, we get:

$$k \geq \frac{\phi_i - \phi_j}{\gcd(T_i, T_j)} \Rightarrow k = \left\lceil \frac{\phi_i - \phi_j}{\gcd(T_i, T_j)} \right\rceil$$
Key observation

- The distance between any arrival of task $\tau_i$ and any arrival of task $\tau_j$ is:

  \[ a_{j,k_1} - a_{i,k_2} = \phi_j + k_1 T_j - \phi_i - k_2 T_i = \phi_j - \phi_i + k(gcd(T_i, T_j)) \]

- Imposing that the difference must not be negative, and $k$ must be integer, we get:

  \[ k \geq \frac{\phi_i - \phi_j}{gcd(T_i, T_j)} \Rightarrow k = \left\lceil \frac{\phi_i - \phi_j}{gcd(T_i, T_j)} \right\rceil \]

- The minimum distance is:

  \[ \Delta_{i,j} = \phi_j - \phi_i + \left\lceil \frac{\phi_i - \phi_j}{gcd(T_i, T_j)} \right\rceil \cdot gcd(T_i, T_j) \]
Observations

- From the formula we can derive the following observations:
  - The value of $\Delta_{i,j}$ is an integer in interval $[0, \gcd(T_i, T_j) - 1]$
  - If $T_i$ and $T_j$ are prime between them (i.e. $\gcd = 1$), then $\Delta_{i,j} = 0$.

- Now we are ready to explain the basic idea behind the new scheduling analysis methodology.
Basic Idea

- Given an hypothetical interval \([x, y]\)
- Assume task \(\tau_i\) arrival time coincides with \(x\)
Basic Idea

- Given an hypothetical interval \([x, y]\)
- Assume task \(\tau_i\) arrival time coincides with \(x\)
- We “push back” all other tasks until they reach the minimum distance from \(\tau_i\) arrival time
Basic Idea

- Given an hypothetical interval \([x, y]\)
- Assume task \(\tau_i\) arrival time coincides with \(x\)
- We “push back” all other tasks until they reach the minimum distance from \(\tau_i\) arrival time
  - there is no need to push it back further (it would be too pessimistic!)
- The df in all intervals starting with \(x\) can only increase after the “pushing back”.

Basic Idea

- Given an hypothetical interval $[x, y]$
- Assume task $\tau_i$ arrival time coincides with $x$
- We “push back” all other tasks until they reach the minimum distance from $\tau_i$ arrival time
  - there is no need to push it back further (it would be too pessimistic!)
- The df in all intervals starting with $x$ can only increase after the “pushing back”.
- Therefore, if no deadline is missed in $[x, y]$, then no deadline is missed in any interval of length $(y - x)$. 
Basic Idea

- Given an hypothetical interval \([x, y]\)
- Assume task \(\tau_i\) arrival time coincides with \(x\)
- We “push back” all other tasks until they reach the minimum distance from \(\tau_i\) arrival time
  - there is no need to push it back further (it would be too pessimistic!)
- The df in all intervals starting with \(x\) can only increase after the “pushing back”.
- Therefore, if no deadline is missed in \([x, y]\), then no deadline is missed in any interval of length \((y - x)\).
- We could build such interval by selecting a task \(\tau_i\) to start at the beginning of the interval, and setting the arrival times of the other tasks at their minimum distances
Problem

- We do not know which task to start with in the interval
- Simple solution: just select each task in turn
Example

\[ \tau_1 = (0, 4, 7, 9) \text{ and } \tau_2 = (2, 5, 8, 12) \]

- We select \( \tau_1 \) to start at 0.
Example

- $\tau_1 = (0, 4, 7, 9)$ and $\tau_2 = (2, 5, 8, 12)$

  - We select $\tau_1$ to start at 0.
  - $\tau_2$ starts at

$$\phi_2 - \phi_1 + \left\lfloor \frac{\phi_1 - \phi_2}{T_1 \mod T_2} \right\rfloor (T_1 \mod T_2) = 2 + \left\lfloor \frac{-2}{3} \right\rfloor 3 = 2$$
Example

- \( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)

- Next, we select \( \tau_2 \) to start at 0.
Example

- \( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)

- Next, we select \( \tau_2 \) to start at 0.

- \( \tau_1 \) starts at

\[
\phi_1 - \phi_2 + \left\lfloor \frac{\phi_2 - \phi_1}{T_2 \mod T_1} \right\rfloor (T_2 \mod T_1) = -2 + \left\lfloor \frac{2}{3} \right\rfloor \cdot 3 = 1
\]
Example

- \( \tau_1 = (0, 4, 7, 9) \) and \( \tau_2 = (2, 5, 8, 12) \)

- Next, we select \( \tau_2 \) to start at 0.

- \( \tau_1 \) starts at

\[
\phi_1 - \phi_2 + \left\lfloor \frac{\phi_2 - \phi_1}{T_2 \mod T_1} \right\rfloor (T_2 \mod T_1) = -2 + \left\lfloor \frac{2}{3} \right\rfloor 3 = 1
\]
Example

- $\tau_1 = (0, 4, 7, 9)$ and $\tau_2 = (2, 5, 8, 12)$
- Next, we select $\tau_2$ to start at 0.
- $\tau_1$ starts at

$$\phi_1 - \phi_2 + \left\lfloor \frac{\phi_2 - \phi_1}{T_2 \mod T_1} \right\rfloor (T_2 \mod T_1) = -2 + \left\lfloor \frac{2}{3} \right\rfloor 3 = 1$$
Main theorem

Given an asynchronous task set $\mathcal{T}$

Let $\mathcal{T}_i'$ be the task set obtained by

- fixing the offset of $\tau_i$ at 0
- setting the offset of all other tasks at their minimum distance from $\tau_i$
Main theorem

- Given an asynchronous task set $\mathcal{T}$
- Let $\mathcal{T}_i'$ be the task set obtained by
  - fixing the offset of $\tau_i$ at 0
  - setting the offset of all other tasks at their minimum distance from $\tau_i$

**Theorem (Pellizzoni and Lipari, ECRTS ’04)**

Given task set $\mathcal{T}$ with $U \leq 1$, scheduled on a single processor, if $\forall 1 \leq i \leq N$ all deadlines in task set $\mathcal{T}_i'$ are met until the first idle time, then $\mathcal{T}$ is feasible.
Figure: 10 tasks with periods multiple of 10
Conclusions

- What is this for?
- Feasibility analysis of asynchronous task set is used for:
  - Reduction of output jitter: by setting an offset it is possible to reduce response time and jitter
  - Analysis of distributed transactions (i.e. chains of tasks related by precedence constraints).
Conclusions

- What is this for?

- Feasibility analysis of asynchronous task set is used for:
  - Reduction of output jitter: by setting an offset it is possible to reduce response time and jitter
  - Analysis of distributed transactions (i.e. chains of tasks related by precedence constraints).

- In both cases, the analysis must be iteratively repeated many times with different offsets;
Conclusions

• What is this for?
• Feasibility analysis of asynchronous task set is used for:
  • Reduction of output jitter: by setting an offset it is possible to reduce response time and jitter
  • Analysis of distributed transactions (i.e. chains of tasks related by precedence constraints).
• in both cases, the analysis must be iteratively repeated many times with different offsets;
• hence we need an efficient analysis (even though it is only sufficient)
References I

M. L. Dertouzos
Control Robotics: The Procedural Control of Physical Processes
Information Processing, 1974

J.Y.-T. Leung and M.L. Merril,
A Note on Preemptive Scheduling of Periodic Real-Time Tasks

S.K. Baruah, L.E. Rosier and R.R. Howell,
Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor
References II

R. Pellizzoni and G. Lipari
Feasibility Analysis of Real-Time Periodic Tasks with Offsets
Real-Time Systems Journal, 2005