Multiprocessor Real-Time Scheduling

Embedded System Software Design

Prof. Ya-Shu Chen
National Taiwan University of Science and Technology

Outline

- Multiprocessor Real-Time Scheduling
- Global Scheduling
- Partitioned Scheduling
- Semi-partitioned Scheduling

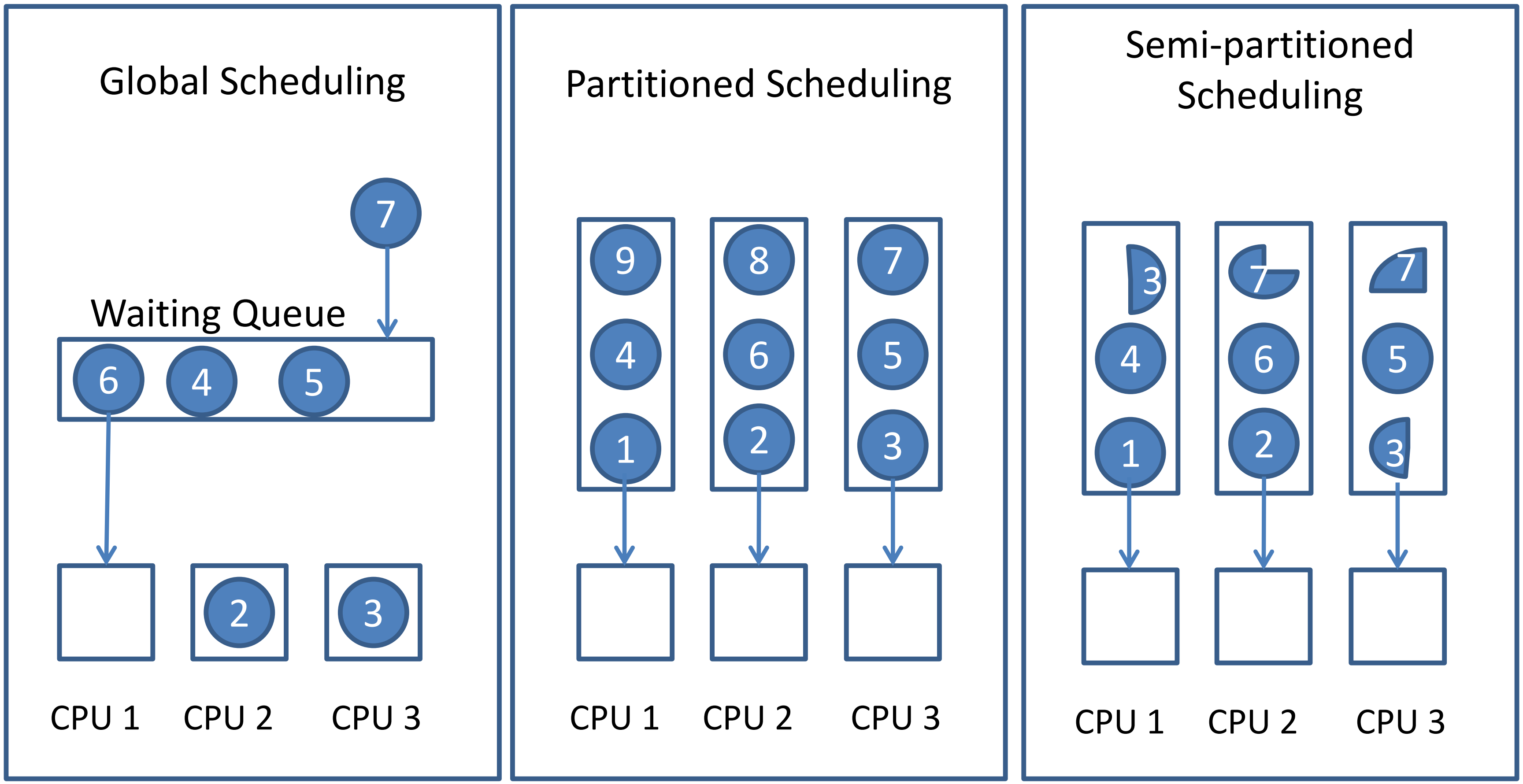
Multiprocessor Models

- Identical (Homogeneous): All the processors have the same characteristics, i.e., the execution time of a job is independent on the processor it is executed.
- Uniform: Each processor has its own speed, i.e., the execution time of a job on a processor is proportional to the speed of the processor.
  - A faster processor always executes a job faster than slow processors do.
  - For example, multiprocessors with the same instruction set but with different supply voltages/frequencies.
- Unrelated (Heterogeneous): Each job has its own execution time on a specified processor
  - A job might be executed faster on a processor, but other jobs might be slower on that processor.
  - For example, multiprocessors with different instruction sets.

Scheduling Models

- Global Scheduling:
  - A job may execute on any processor.
  - The system maintains a global ready queue.
  - Execute the M highest-priority jobs in the ready queue, where M is the number of processors.
  - It requires high on-line overhead.
- Partitioned Scheduling:
  - Each task is assigned on a dedicated processor.
  - Schedulability is done individually on each processor.
  - It requires no additional on-line overhead.
- Semi-partitioned Scheduling:
  - Adopt task partitioning first and reserve time slots (bandwidths) for tasks that allow migration.
  - It requires some on-line overhead.

Scheduling Models



Global Scheduling

- All ready tasks are kept in a global queue
- A job can be migrated to any processor.
- Priority-based global scheduling:
  - Among the jobs in the global queue, the M highest priority jobs are chosen to be executed on M processors.
  - Task migration here is assumed with no overhead.
- Global-EDF: When a job finishes or arrives to the global queue, the M jobs in the queue with the shortest absolute deadlines are chosen to be executed on M processors.
- Global-RM: When a job finishes or arrives to the global queue, the M jobs in the queue with the highest priorities are chosen to be executed on M processors.

Global Scheduling

- Advantages:
  - Effective utilization of processing resources (if it works)
  - Unused processor time can easily be reclaimed at run-time (mixture of hard and soft RT tasks to optimize resource utilization)
- Disadvantages:
  - Adding processors and reducing computation times and other parameters can actually decrease optimal performance in some scenarios!
  - Poor resource utilization for hard timing constraints
  - Few results from single-processor scheduling can be used

Schedule Anomaly

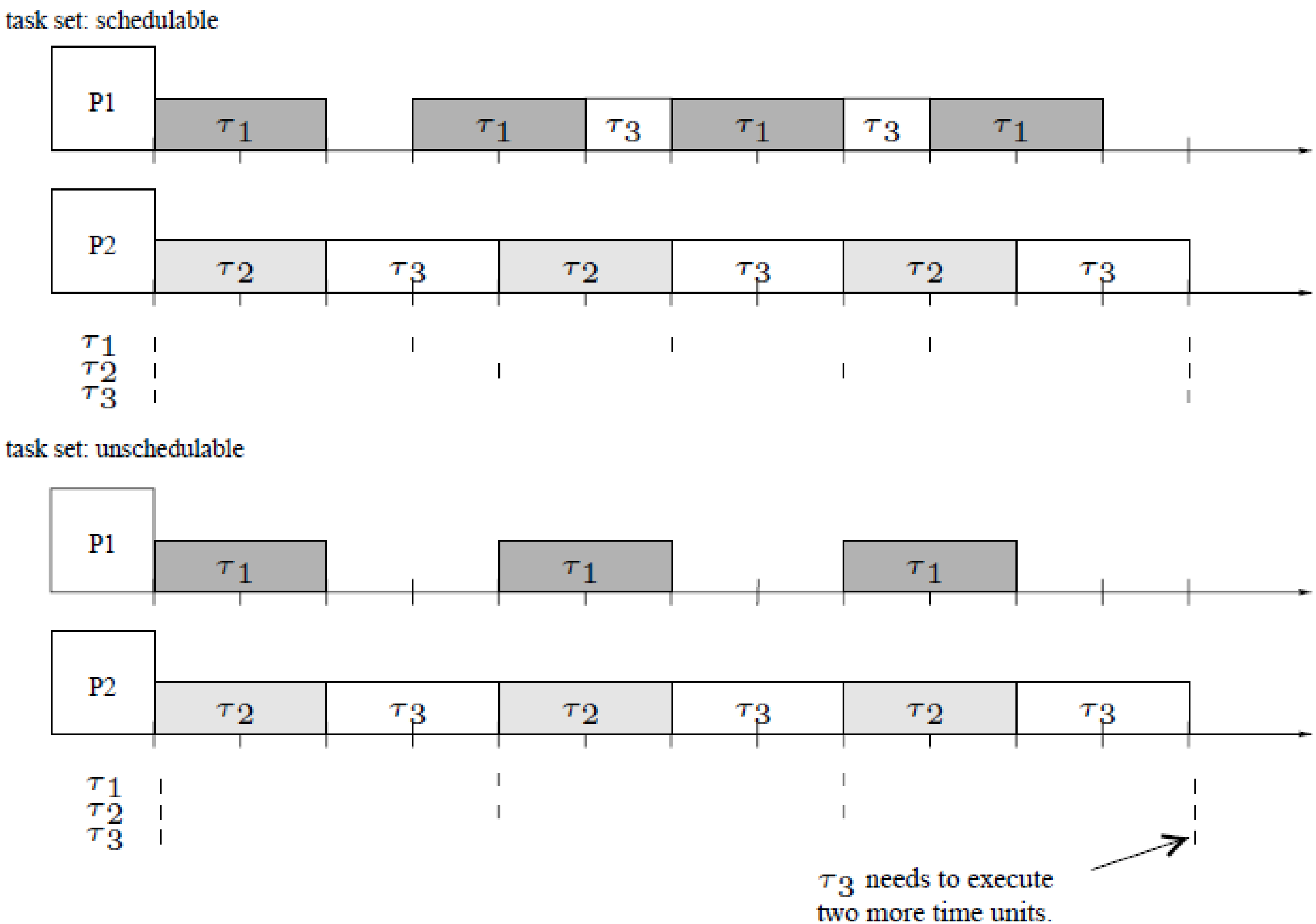
- **Anomaly 1**

  A decrease in processor demand from higher-priority tasks can *increase* the interference on a lower-priority task because of the change in the time when the tasks execute

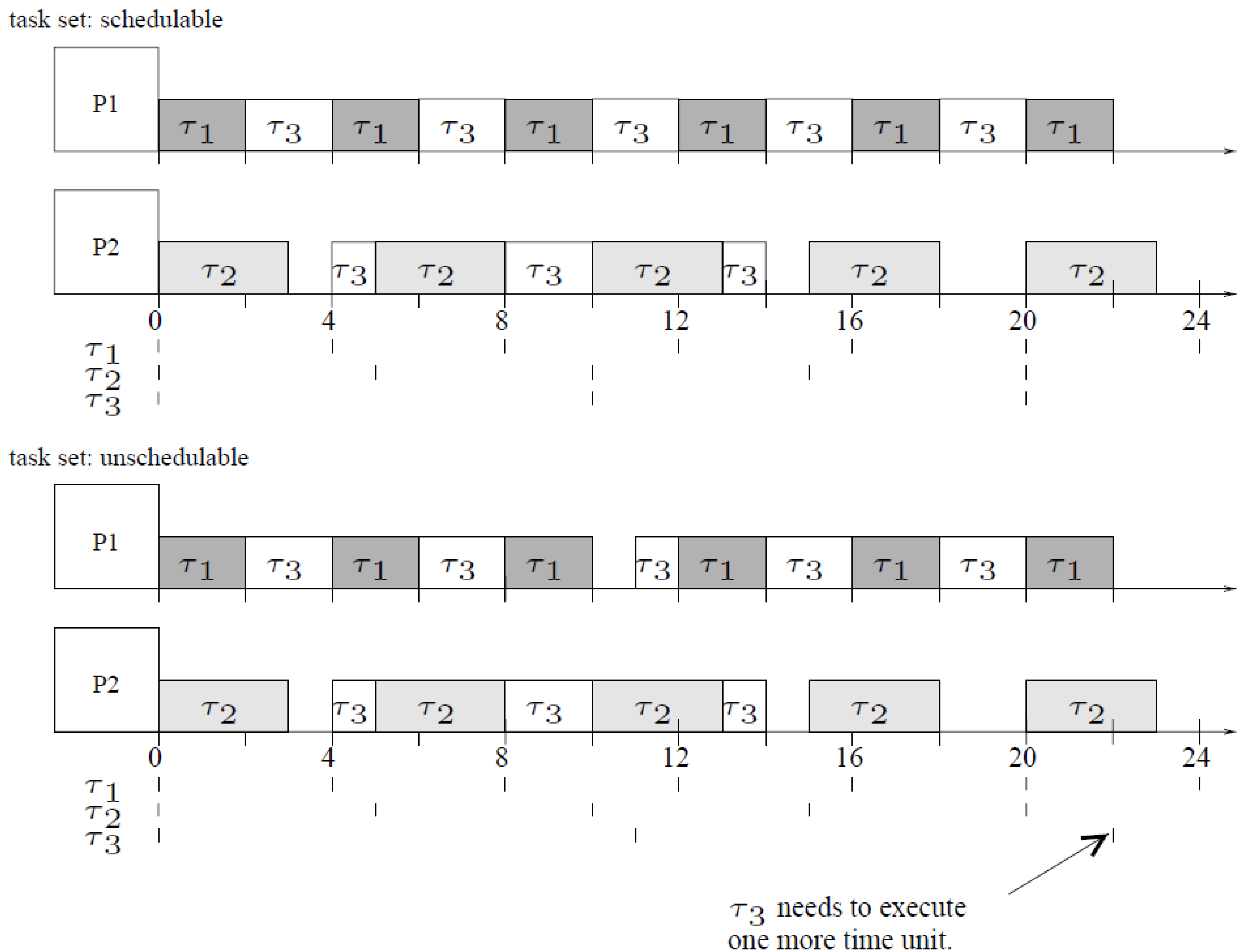
- **Anomaly 2**

  A decrease in processor demand of a task *negatively* affects the task itself because the change in the task arrival times make it suffer more interference

Anomaly 1


task set: schedulable
P1
τ_1
τ_1
τ_3
τ_1
τ_3
τ_1
P2
τ_2
τ_3
τ_2
τ_3
τ_2
τ_3
τ_1
τ_2
τ_3
task set: unschedulable
P1
τ_1
τ_1
τ_1
P2
τ_2
τ_3
τ_2
τ_3
τ_2
τ_3
τ_1
τ_2
τ_3
τ_3 needs to execute two more time units.

Anomaly 2


task set: schedulable
P1
τ_1 τ_3 τ_1 τ_3 τ_1 τ_3 τ_1 τ_3 τ_1 τ_3 τ_1
P2
τ_2 τ_3 τ_2 τ_3 τ_2 τ_3 τ_2 τ_2
0 4 8 12 16 20 24
τ_1
τ_2
τ_3
task set: unschedulable
P1
τ_1 τ_3 τ_1 τ_3 τ_1 τ_3 τ_1 τ_3 τ_1 τ_3 τ_1
P2
τ_2 τ_3 τ_2 τ_3 τ_2 τ_3 τ_2 τ_2
0 4 8 12 16 20 24
τ_1
τ_2
τ_3
τ_3 needs to execute
one more time unit.

Dhall effect

- Dhall effect : For Global-EDF or Global-RM, the least upper bound for schedulability analysis is at most 1.
- On 2 processors:

Task	T	D	C	U
T1	10	10	5	0.5
T2	10	10	5	0.5
T3	12	12	8	0.67

- T3 is not schedulable

Schedulability Test

- A set of periodic tasks $t_1, t_2, \ldots, t_N$ with implicit deadlines is schedulable on M processors by using preemptive Global EDF scheduling if

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \le M\left(1 - \frac{C_k}{T_k}\right) + \frac{C_k}{T_k},$$

where t_k is the task with the largest utilization C_k/T_k

Weakness of Global Scheduling

- Migration overhead
- Schedule Anomaly

Partitioned Scheduling

- Two steps:
  - Determine a mapping of tasks to processors
  - Perform run-time single-processor scheduling
- Partitioned with EDF
  - Assign tasks to the processors such that no processor’s capacity is exceeded (utilization bounded by 1.0)
  - Schedule each processor using EDF

Bin-packing Problem

Given a bin size V and a list $a_1, \ldots, a_n$ of sizes of the items to pack, find an integer B and a B-partition $S_1 \cup \cdots \cup S_B$ of $\{1, \ldots, n\}$ such that $\sum_{i \in S_k} a_i \leq V$, for all $k = 1, \ldots, B$.
A solution is optimal if it has minimal B.

The problem is NP-complete !!

Bin-packing to Multiprocessor Scheduling

- The problem concerns packing objects of varying sizes in boxes (”bins”) with the objective of minimizing number of used boxes.
  - Solutions (Heuristics): First Fit
- Application to multiprocessor systems:
  - Bins are represented by processors and objects by tasks.
  - The decision whether a processor is ”full” or not is derived from a utilization-based schedulability test.

Partitioned Scheduling

- Advantages:
  - Most techniques for single-processor scheduling are also applicable here
- Partitioning of tasks can be automated
  - Solving a bin-packing algorithm
- Disadvantages:
  - Cannot exploit/share all unused processor time
  - May have very low utilization, bounded by 50%

Partitioned Scheduling Problem

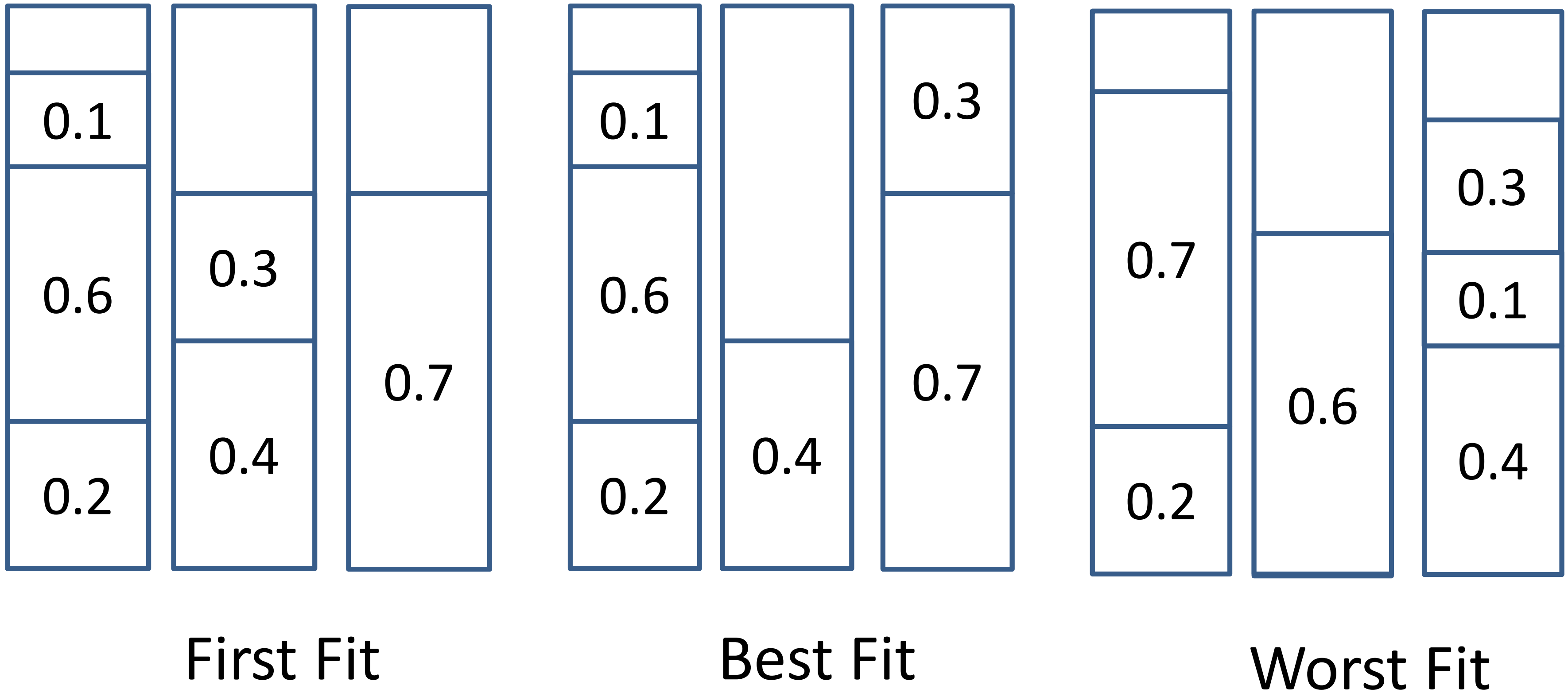
Given a set of tasks with arbitrary deadlines, the objective is to decide a feasible task assignment onto M processors such that all the tasks meet their timing constraints, where C_i is the execution time of task t_i on any processor m.

Partitioned Algorithm

- First-Fit: choose the one with the smallest index
- Best-Fit: choose the one with the maximal utilization
- Worst-Fit: choose the one with the minimal utilization

Partitioned Example

- 0.2 -> 0.6 -> 0.4 -> 0.7 -> 0.1 -> 0.3



EDF with First Fit

Input: A task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a set of processors $\{p_1, \ldots, p_m\}$
Output: j; number of processors required.

$i \coloneqq 1; j \coloneqq 1;\ k_q = 0;\ (\forall_q)$
while $(i \le n)$ **do**
    $q \coloneqq 1;$
    while $((U_q + u_i) > 1)$ **do**
        $q \coloneqq q + 1;$ /* increase the processor index */
    $U_q := U_q + u_i;\ k_q := k_q + 1;$
    if $(q > j)$ **then**
        j $\coloneqq q;$
    $i \coloneqq i + 1;$
return (j);
end

Schedulability Test

Lopez [3] proves that the worst-case achievable utilization for EDF scheduling and FF allocation (EDF-FF) takes the value

If all the tasks have an utilization factor C/T under a value α, where m is the number of processors

$$\boxed{U_{wc}^{EDF-FF}(m,\beta)=\frac{\beta m+1}{\beta+1}}$$ where $\beta=\lfloor 1/\alpha \rfloor$

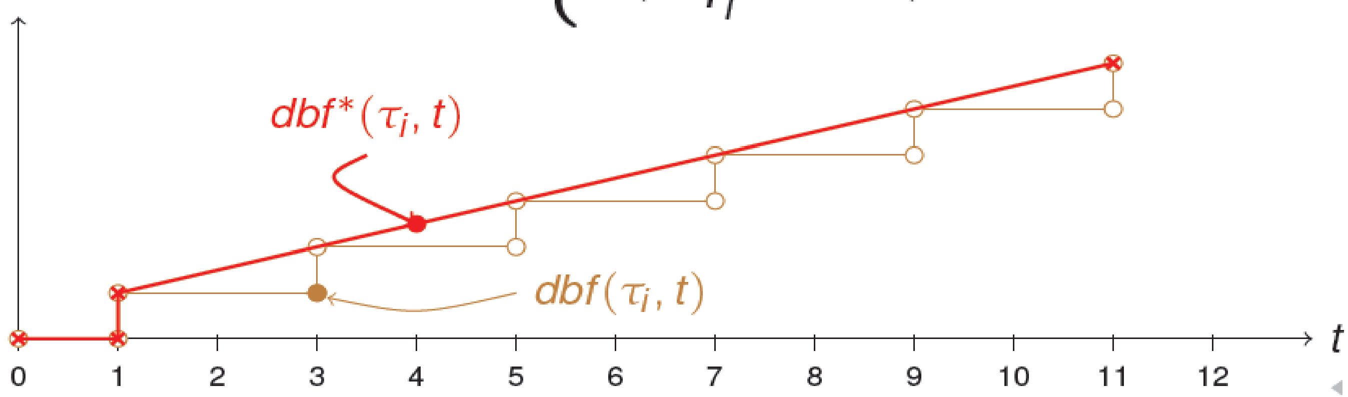
Demand Bound Function

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

$$dbf(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} C_i.$$

- We need approximation to enforce polynomial-time schedulability test

$$dbf^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i \\ \left(\frac{t-D_i}{T_i} + 1\right) C_i & \text{otherwise.} \end{cases}$$



Deadline Monotonic Partition

Input: $\mathbf{T}, M$;

re-index (sort) tasks such that $D_i \leq D_j$ for $i < j$;

$\mathbf{T}_i \leftarrow \emptyset, U_i \leftarrow 0, \forall m = 1, 2, \ldots, M$;

for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**

    for $m = 1$ to M **do**

        if $\frac{C_i}{T_i} + \sum_{\tau_j \in \mathbf{T}_m} \frac{C_j}{T_j} \leq 1$ and $C_i + \sum_{\tau_j \in \mathbf{T}_m} dbf^*(\tau_j, D_i) \leq D_i$ **then**

            assign task τ_i onto processor m and $\mathbf{T}_m \leftarrow \mathbf{T}_m \cup \{\tau_i\}$;

            break;

    if τ_i is not assigned **then**

        return "The task assignment fails";

return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

Schedulabiliy Test

Theorem 4 *Any sporadic task system τ is successfully scheduled by Algorithm* PARTITION *on m unit-capacity processors, for any*

$$m \geq \left(\frac{2\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}} + \frac{u_{\text{sum}} - u_{\max}}{1 - u_{\max}} \right) \qquad (14)$$

$$\delta_{\max} \stackrel{\text{def}}{=} \max_{i=1}^{n} (e_i/d_i) \qquad u_{\max} \stackrel{\text{def}}{=} \max_{i=1}^{n} (u_i)$$

$$\delta_{\text{sum}} \stackrel{\text{def}}{=} \max_{t>0} \left(\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, t)}{t} \right) \qquad u_{\text{sum}} \stackrel{\text{def}}{=} \sum_{j=1}^{n} u_j$$

Weakness of Partitioned Scheduling

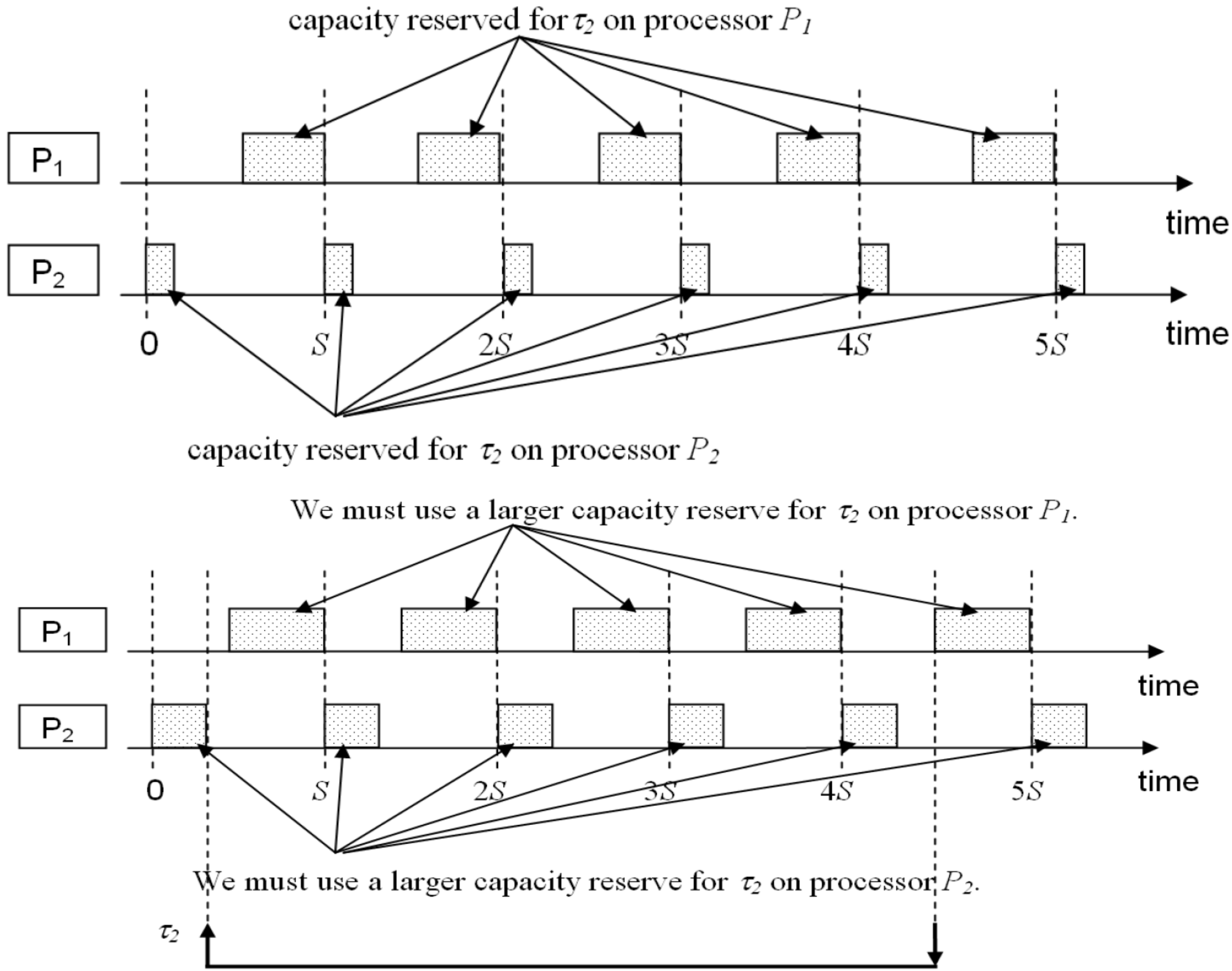
- Restricting a task on a processor reduces the schedulability
- Restricting a task on a processor makes the problem NP-hard
- Example: Suppose that there are M processors and M + 1 tasks with the same period T and the (worst-case) execution times of all these M + 1 tasks are T/2 + *e* with *e* > 0
  - With partitioned scheduling, it is not schedulable

Semi-partitioned Scheduling

- Tasks are first partitioned into processor.
- To reduce the utilization, we again pick the processor with the minimum task utilization
- If a task cannot fit into the picked processor, we will have to split it into multiple (two or even more) parts.
- If t_i is split and assigned to a processor m and the utilization on processor m after assigning t_i is at most U(scheduler,N), then t_i is so far schedulable.

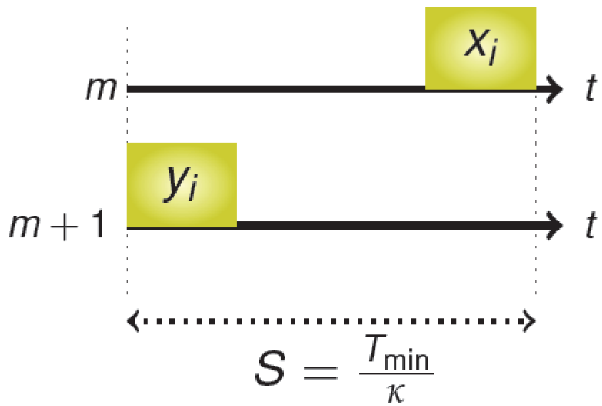
Semi-partitioned EDF

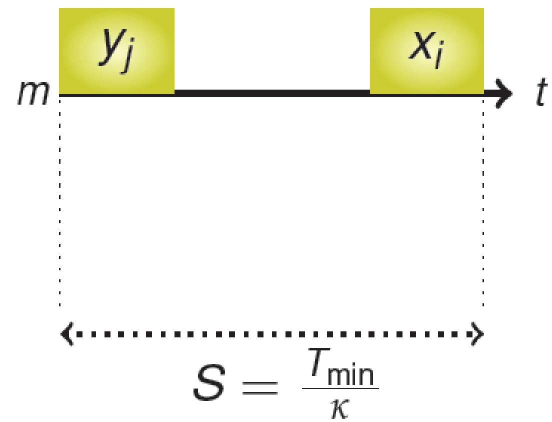
- T_{min} is the minimum period among all the tasks.
- By a user-designed parameter *k*, we divide time into slots with length $S = T_{min}/k$.
- We can use the first-fit approach by splitting a task into 2 subtasks, in which one is executed on processor m and the other is executed on processor m + 1.
- Execution of a split task is only possible in the reserved time window in the time slot.
- Applying first-fit algorithm, by taking SEP as the upper bound of utilization on a processor.
- If a task does not fit, split this task into two subtasks and allocate a new processor, one is assigned on the processor under consideration, and the other is assigned on the newly allocated processor.


capacity reserved for τ_2 on processor P_1
P_1
P_2
time
time
0
S
$2S$
$3S$
$4S$
$5S$
capacity reserved for τ_2 on processor P_2
We must use a larger capacity reserve for τ_2 on processor P_1.
P_1
P_2
time
time
0
S
$2S$
$3S$
$4S$
$5S$
We must use a larger capacity reserve for τ_2 on processor P_2.
τ_2

Semi-partitioned EDF

- For each time slot, we will reserve two parts.





If a task t_i is split, the task can be served only within these two pre-defined time slots with length x_i and y_i .

A processor can host two split tasks, t_i and t_j. t_i is served at the beginning of the time slot, and t_j is served at the end.

The schedule is EDF, but if a split task instance is in the ready queue, it is executed in the reserved time region.

Semi-partitioned EDF

- We can assign all the tasks t_i with U_i > SEP on a dedicated processor. So, we only consider tasks with U_i no larger SEP.

$m \leftarrow 1, U_m \leftarrow 0;$
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
    if $\frac{C_i}{T_i} + U_m \leq SEP$ **then**
        assign task τ_i on processor m;
        $U_m \leftarrow U_m + \frac{C_i}{T_i}$;
    else
        assign task τ_i on processor m with $lo\_split(\tau_i)$ set to $SEP - U_m$ and on processor $m + 1$ with $high\_split(\tau_i)$ set to $\frac{C_i}{T_i} - (SEP - U_m)$;
        $m \leftarrow m + 1$ and $U_m \leftarrow \frac{C_i}{T_i} - (SEP - U_m)$;

When executing, the reservation to serve t_i is to set
x_i to S X (f + lo_split(t_i)) and y_i to S X (f + high_split(t_i)).
SEP is set as a constant.

Two Split Tasks on a Processor

- For split tasks to be schedulable, the following sufficient conditions have to be satisfied
  - lo_split(t_i) + f + high_split(t_i) + f <= 1 for any split task t_i .
  - lo_split(t_j) + f + high_split(t_i) + f <= 1 when t_i and t_j are assigned on the same processor.
- Therefore, the "magic value" SEP

$$SEP \leq 1 - 2f \leq 1 - 2(\sqrt[2]{\kappa(\kappa+1)} - \kappa).$$

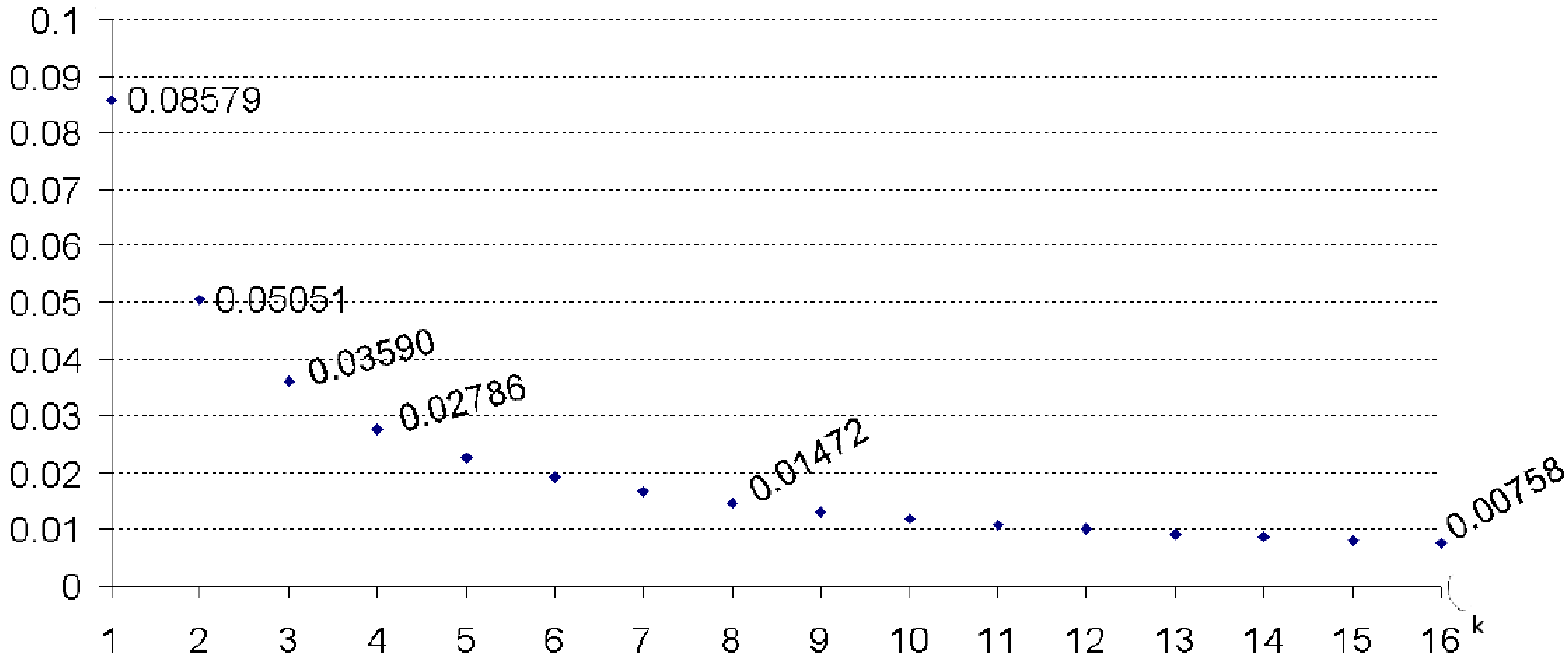
- However, we still have to guarantee the schedulability of the non-split tasks. It can be shown that the sufficient condition is

$$SEP \leq 1 - 4f \leq 1 - 4(\sqrt[2]{\kappa(\kappa+1)} - \kappa).$$

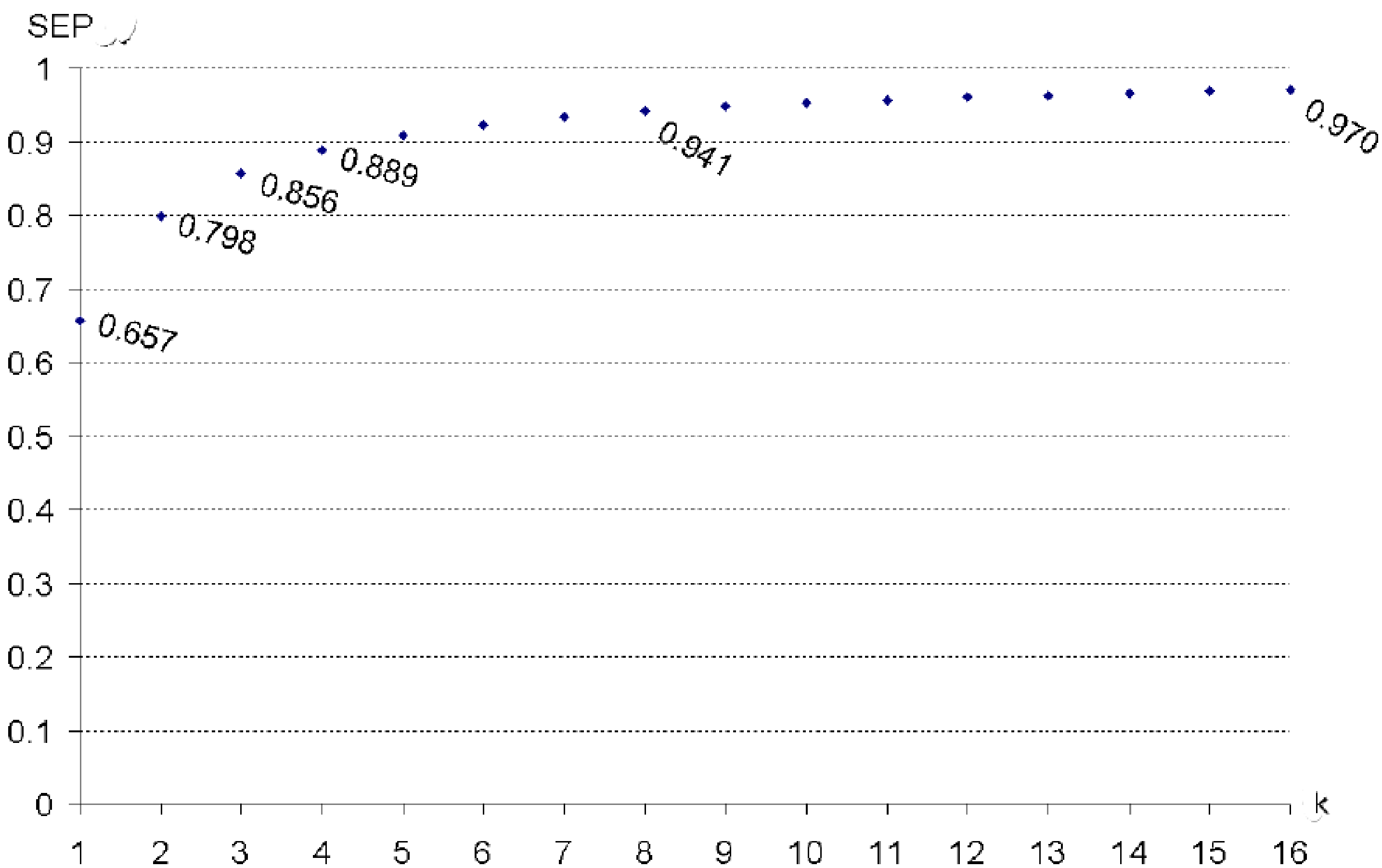
Schedulability Test

By taking SEP as $1-4(\sqrt[2]{\kappa(\kappa+1)}-\kappa)$ and $f=\sqrt[2]{\kappa(\kappa+1)}-\kappa$, the above algorithm guarantees to derive feasible schedule if $\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \leq M' \cdot SEP$ and $\frac{C_i}{T_i} \leq SEP$ for all tasks τ_i.

Magic Values: f



Magic Values: SEP



Reference

- Multiprocessor Real-Time Scheduling
  - Dr. Jian-Jia Chen: Multiprocessor Scheduling. Karlsruhe Institute of Technology (KIT): 2011-2012
- Global Scheduling
  - Sanjoy K. Baruah: Techniques for Multiprocessor Global Schedulability Analysis. RTSS 2007: 119-128
- Partitioned Scheduling
  - Sanjoy K. Baruah, Nathan Fisher: The Partitioned Multiprocessor Scheduling of Sporadic Task Systems. RTSS 2005: 321-329
- Semi-partitioned Scheduling
  - Björn Andersson, Konstantinos Bletsas: Sporadic Multiprocessor Scheduling with Few Preemptions. ECRTS 2008: 243-252

See You Next Week