Multiprocessor Real-Time Scheduling: A Summary

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

08 Jan. 2019

Outline

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

- Partitioned Scheduling:
  - Each task is assigned on a dedicated processor.
  - Schedulability is done individually on each processor.
  - It requires no additional on-line overhead.
- 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.

Problem Definition: Partitioned Scheduling

Partitioned Scheduling

Given a set $\mathbf{T}$ of tasks with implicit deadlines, i.e., $\forall \tau_i \in \mathbf{T}$, $T_i = D_i$, the objective is to decide a feasible task assignment onto M processors such that all the tasks meet their timing constraints, where C_{im} is the execution time of task τ_i on processor m.

- For identical multiprocessors: $C_i = C_{i1} = C_{i2} = \cdots = C_{iM}$.
- For uniform multiprocessors: each processor m has a speed s_m, in which $C_{im}s_m$ is a constant.
- For unrelated multiprocessors: C_{im} is an independent parameter.

Hardness and Approximation of Partitioned Scheduling

$\mathcal{N}P$-complete

Deciding whether there exists a feasible task assignment is $\mathcal{N}P$-complete in the strong sense.

Proof

Reduced from the 3-Partition problem.

Hardness and Approximation of Partitioned Scheduling

$\mathcal{N}P$-complete

Deciding whether there exists a feasible task assignment is $\mathcal{N}P$-complete in the strong sense.

Proof

Reduced from the 3-Partition problem.

- Approximations are possible, but what do we approximate when only binary decisions (Yes or No) have to be made?
  - Deadline relaxation: requires modifications of task specification
  - Period relaxation: requires modifications of task specification
  - Resource augmentation by speeding up: requires a faster platform
  - Resource augmentation by allocating more processors: requires a better platform

Approximation Algorithms

An algorithm $\mathcal{A}$ is called an η-approximation algorithm (for a minimization problem) if it guarantees to derive a feasible solution for any input instance I with at most η times of the objective function of an optimal solution. That is,

$$\mathcal{A}(I) \leq \eta OPT(I),$$

where $OPT(I)$ is the objective function of an optimal solution.

Terminologies Used in Scheduling Theory

Graham's Scheduling Algorithm Classification

- Classification: $a|b|c$
  - a: machine environment
    (e.g., uniprocessor, multiprocessor, distributed, ...)
  - b: task and resource characteristics
    (e.g., preemptive, independent, synchronous, ...)
  - c: performance metric and objectives
    (e.g., $L_{\max}$, sum of finish times, ...)
- Makespan problem:
  - $M||C_{\max}$
  - Input: M identical processors and N jobs with given execution times arriving at time 0
  - Output: Assign a job to a processor and execute the jobs to minimize the maximum completion time

Bin Packing Problem

- Given a bin size b, and a set of items with individual sizes, the objective is to assign each item to a bin without violating the bin size constraint such that the number of allocated bins is minimized.

Outline

Largest-Utilization-First (LUF) - for EDF Scheduling

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

re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
$\mathbf{T}_m \leftarrow \emptyset, U_m \leftarrow 0, \forall m = 1, 2, \ldots, M$;
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
    find m^* with the minimum utilization, i.e., $U_{m^*} = \min_{m \leq M} U_m$;
    if $U_{m^*} + \frac{C_i}{T_i} > 1$ **then**
        return "The task assignment fails";
    else
        assign task τ_i onto processor m^*, where
        $U_{m^*} \leftarrow U_{m^*} + \frac{C_i}{T_i}, \mathbf{T}_{m^*} \leftarrow \mathbf{T}_{m^*} \cup \{\tau_i\}$;
return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

Largest-Utilization-First (LUF) - for EDF Scheduling

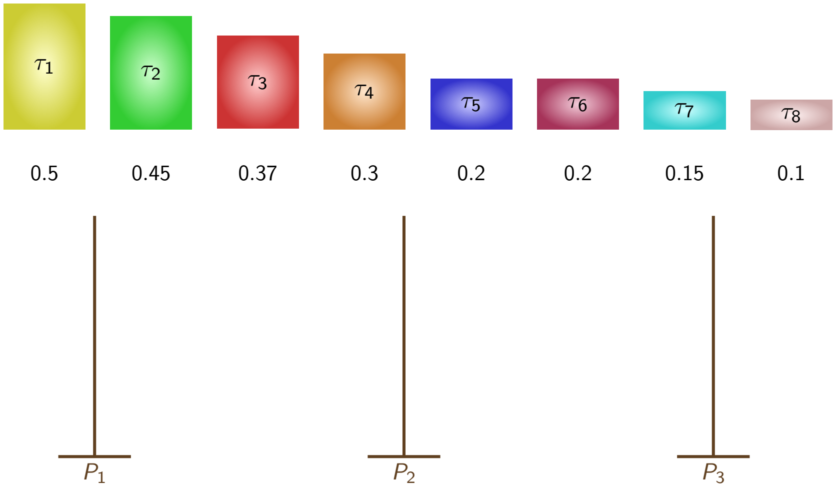
Input: $\mathbf{T}, M$;

re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
$\mathbf{T}_m \leftarrow \emptyset, U_m \leftarrow 0, \forall m = 1, 2, \ldots, M$;
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
    find m^* with the minimum utilization, i.e., $U_{m^*} = \min_{m \leq M} U_m$;
    if $U_{m^*} + \frac{C_i}{T_i} > 1$ **then**
        return "The task assignment fails";
    else
        assign task τ_i onto processor m^*, where
        $U_{m^*} \leftarrow U_{m^*} + \frac{C_i}{T_i}, \mathbf{T}_{m^*} \leftarrow \mathbf{T}_{m^*} \cup \{\tau_i\}$;
return feasible task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_M$;

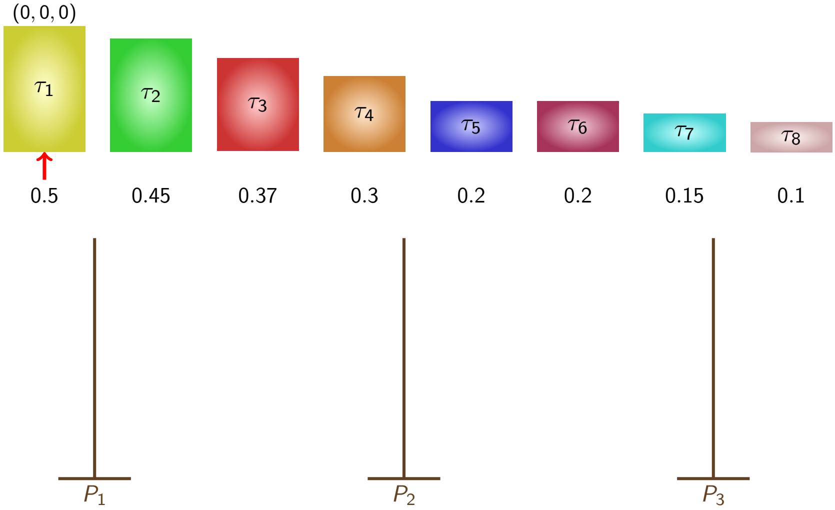
Properties

- The time complexity is $O((N + M)\log(N + M))$
- If a solution is derived, the task assignment is feasible by using EDF.

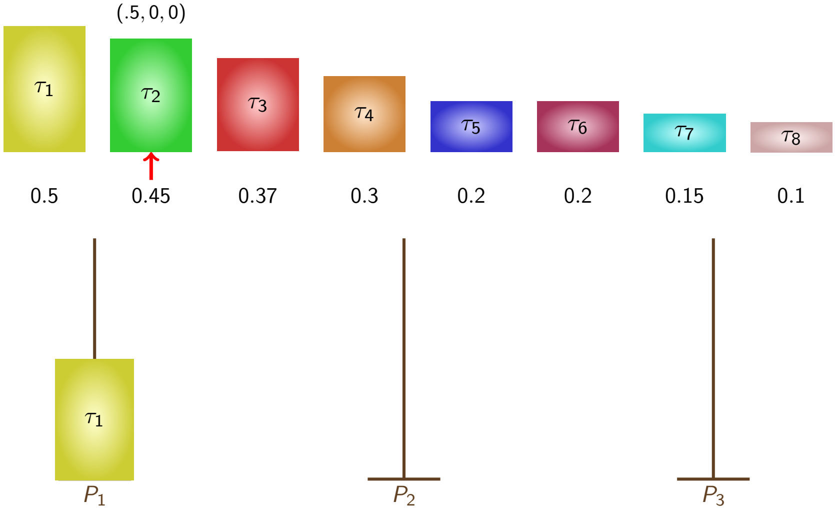
Algorithm LUF



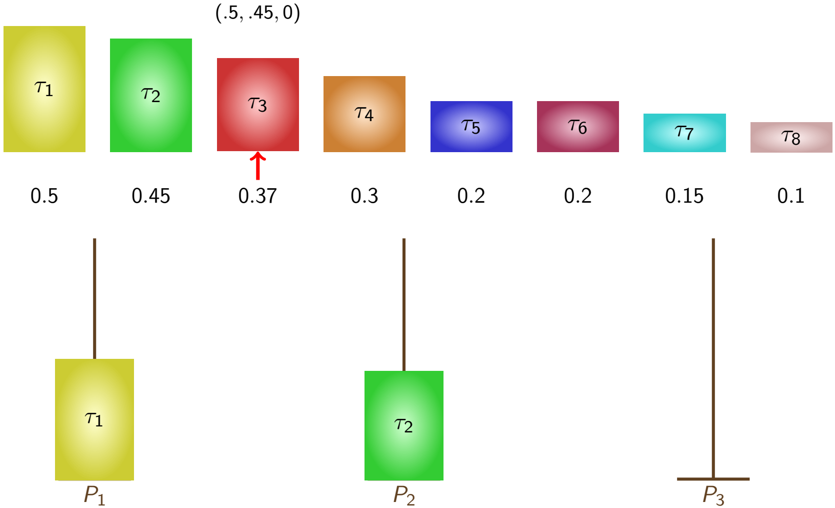
Algorithm LUF



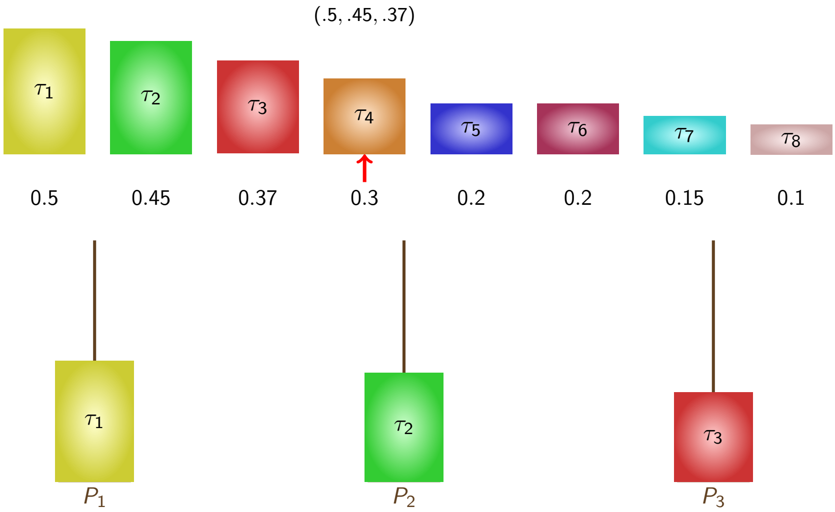
Algorithm LUF



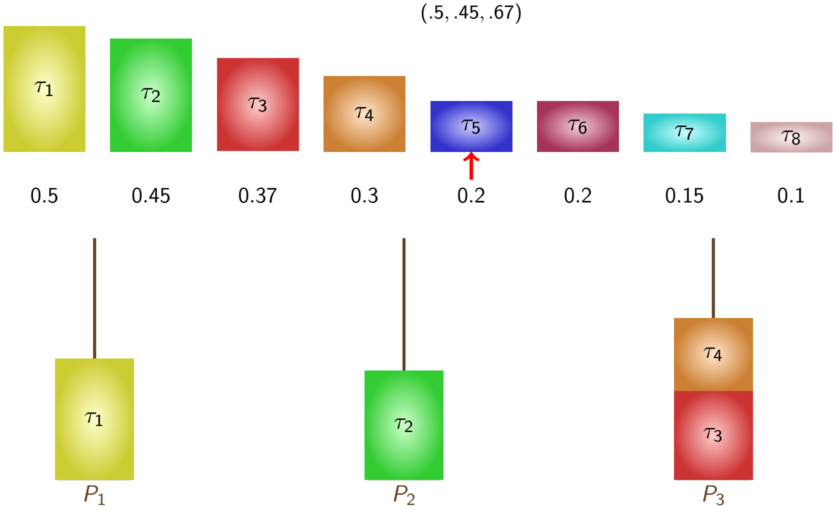
Algorithm LUF



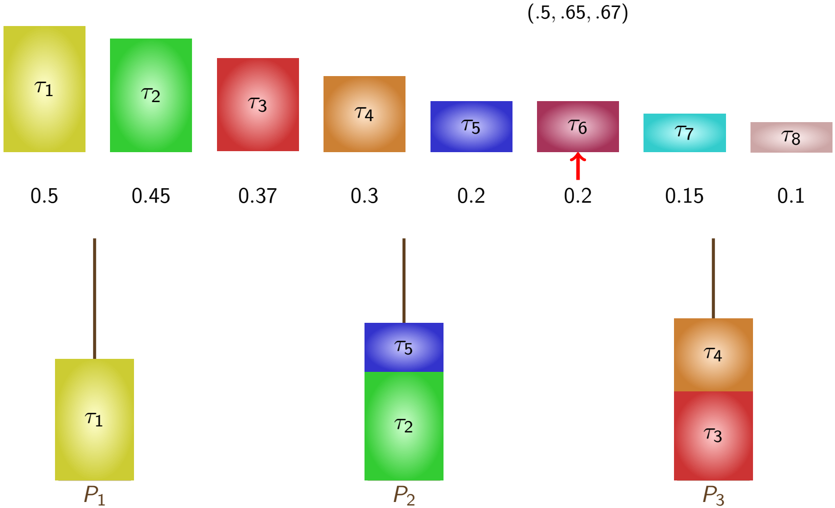
Algorithm LUF



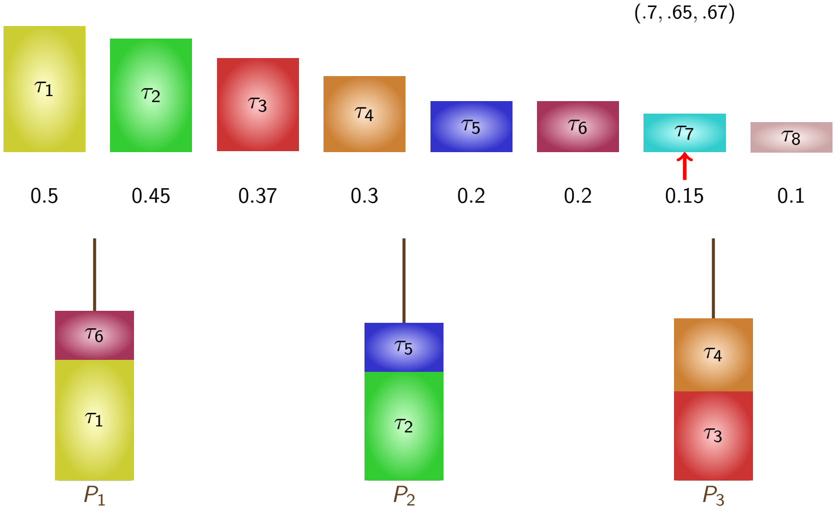
Algorithm LUF



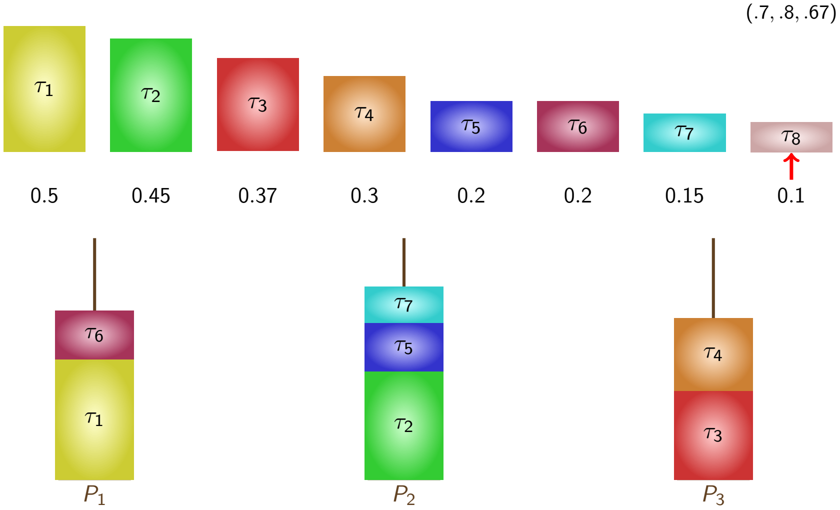
Algorithm LUF



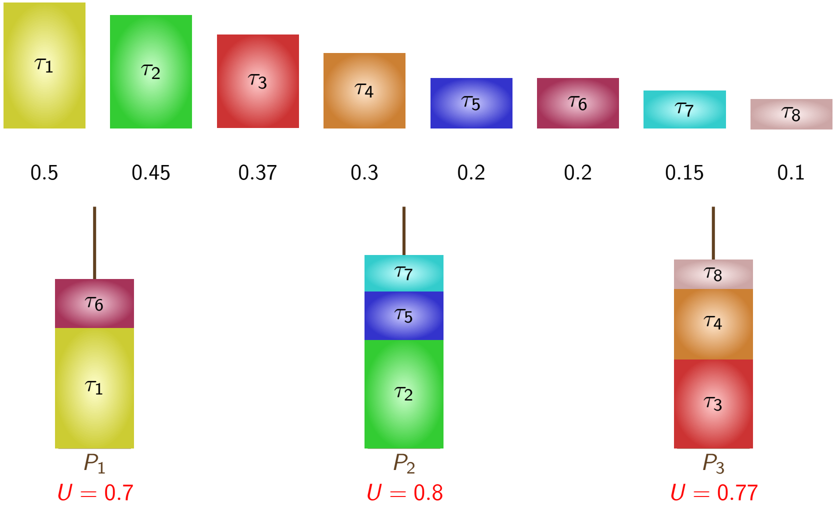
Algorithm LUF



Algorithm LUF



Algorithm LUF



Optimality of Algorithm LUF

Theorem

If an optimal assignment for minimizing the maximal utilization results in at most two tasks on any processor, LUF is optimal.

Proof

The proof is omitted.

What Happens if Algorithm LUF Fails?

Assume that there exists a feasible task partition on M processors (for providing the analysis of resource augmentation).

- Suppose that Algorithm LUF fails when assigning task τ_j and U_m for $m = 1, 2, \ldots, M$ is the utilization of processor m before assigning τ_j.
- Let U_{opt} be the utilization of the optimal assignment for minimizing the maximal utilization for tasks $\{\tau_1, \tau_2, \ldots, \tau_j\}$.
- By definition, $1 \geq U_{opt} \geq \sum_{i=1}^{j} \frac{C_i/T_i}{M}$.
- $\frac{C_j}{T_j} \leq \frac{1}{3} U_{opt}$: otherwise, there will be at most two tasks on any processors in the optimal solution. $\Rightarrow$ this contradicts the assumption that Algorithm LUF fails as it is optimal.
- Since $U_{m^*} \leq U_m$, we know that $U_{m^*} \leq \sum_{m=1}^{M} \frac{U_m}{M} = \sum_{i=1}^{j-1} \frac{C_i/T_i}{M}$.
- Therefore,

$$\frac{C_j}{T_j} + U_{m^*} \leq \frac{C_j}{T_j}(1 - \frac{1}{M}) + \sum_{i=1}^{j} \frac{C_i/T_i}{M} \leq \left(\frac{4}{3} - \frac{1}{3M}\right) U_{opt} \leq \left(\frac{4}{3} - \frac{1}{3M}\right).$$

Algorithm LUF^+: Resource Augmentation on Processors

Input: $\mathbf{T}$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_1 \leftarrow \emptyset, U_1 \leftarrow 0, \hat{M} \leftarrow 1$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;
5: **if** no such a processor exists **then**
6: $\hat{M} \leftarrow \hat{M} + 1, \mathbf{T}_{\hat{M}} \leftarrow \emptyset, U_{\hat{M}} \leftarrow 0$;
7: $m^* \leftarrow \hat{M}$;
8: assign task τ_i onto processor m^*, where $U_i \leftarrow U_i + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;
9: return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{\hat{M}}$;

Algorithm LUF^+: Resource Augmentation on Processors

Input: $\mathbf{T}$;

```
re-index (sort) tasks such that C_i/T_i >= C_j/T_j for i < j;
T_1 <- ∅, U_1 <- 0, M̂ <- 1;
for i = 1 to N, where N = |T| do
    find a processor m* with U_{m*} + C_i/T_i <= 1;
    if no such a processor exists then
        M̂ <- M̂ + 1, T_M̂ <- ∅, U_M̂ <- 0;
        m* <- M̂;
    assign task τ_i onto processor m*, where
    U_i <- U_i + C_i/T_i, T_i <- T_i ∪ {τ_i};
return task assignment T_1, T_2, ..., T_M̂;
```

re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;

$\mathbf{T}_1 \leftarrow \emptyset, U_1 \leftarrow 0, \hat{M} \leftarrow 1$;

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

  find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

  if no such a processor exists **then**

    $\hat{M} \leftarrow \hat{M} + 1, \mathbf{T}_{\hat{M}} \leftarrow \emptyset, U_{\hat{M}} \leftarrow 0$;

    $m^* \leftarrow \hat{M}$;

  assign task τ_i onto processor m^*, where $U_i \leftarrow U_i + \frac{C_i}{T_i}, \mathbf{T}_i \leftarrow \mathbf{T}_i \cup \{\tau_i\}$;

return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{\hat{M}}$;

Properties

- The time complexity is $O(N \log N)$ or $O(N^2)$, depending on the fitting approaches.
- The resulting solution is feasible on $\hat{M}$ processors.

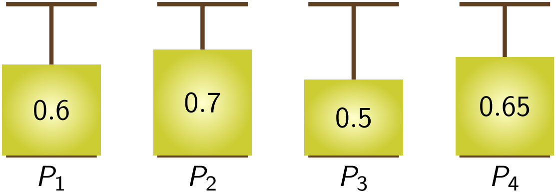
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

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

Suppose that we want to assign a task with utilization equal to 0.1.



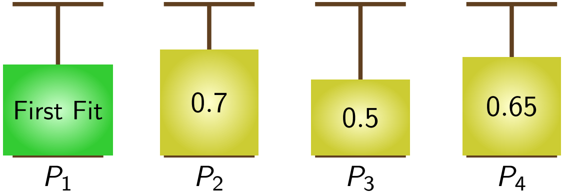
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

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

Suppose that we want to assign a task with utilization equal to 0.1.



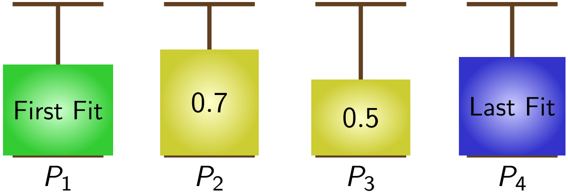
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

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

Suppose that we want to assign a task with utilization equal to 0.1.



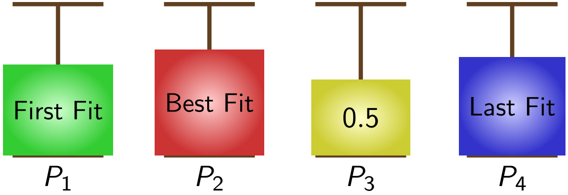
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

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

Suppose that we want to assign a task with utilization equal to 0.1.



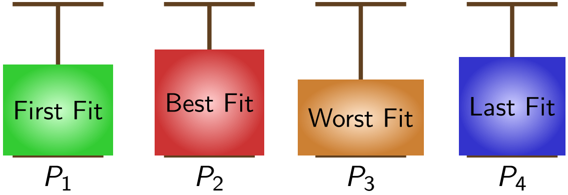
Different Fitting Approaches

4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq 1$;

Fitting Strategies

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

Suppose that we want to assign a task with utilization equal to 0.1.



Algorithm LUF^+: How Many Processors?

- Suppose that the processor used by Algorithm LUF^+ is $\hat{M} \geq 2$.
- Let m^* be the processor with the minimum utilization.
- By the fitting algorithm, we know that $U_m + U_{m^*} > 1$ and $U_m \geq U_{m^*}$ for all the other processors ms.
- If $U_{m^*} \leq 0.5$, by $U_m > 1 - U_{m^*}$, we know that

$$\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \geq U_{m^*} + \sum_{m=1, m \neq m^*}^{\hat{M}} U_m \geq \hat{M} - 1 - (\hat{M} - 2) U_{m^*} \leq (\hat{M} - 2)(1 - U_{m^*}) + 1 \geq \frac{\hat{M}}{2}.$$

- If $U_{m^*} > 0.5$, by $U_m \geq U_{m^*}$, we know that

$$\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \geq U_{m^*} + \sum_{m=1, m \neq m^*}^{\hat{M}} U_m \geq \frac{\hat{M}}{2}.$$

Theorem

Algorithm LUF^+ is a 2-approximation algorithm (with respect to allocating more processors).

Outline

Largest-Utilization-First (LUF^+) - for RM Scheduling

Input: $\mathbf{T}$;

re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
$\mathbf{T}_1 \leftarrow \emptyset, U_1 \leftarrow 0, n_1 \leftarrow 0; \hat{M} \leftarrow 1$;
for $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
    find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq (n_{m^*} + 1)\left(2^{\frac{1}{n_{m^*}+1}} - 1\right)$;
    if no such a processor exists **then**
        $\hat{M} \leftarrow \hat{M} + 1, \mathbf{T}_{\hat{M}} \leftarrow \emptyset, U_{\hat{M}} \leftarrow 0,\ n_{\hat{M}} \leftarrow 0$;
        $m^* \leftarrow \hat{M}$;
    assign task τ_i onto processor m^*, where
    $U_{m^*} \leftarrow U_{m^*} + \frac{C_i}{T_i}, \mathbf{T}_{m^*} \leftarrow \mathbf{T}_{m^*} \cup \{\tau_i\}, n_{m^*} \leftarrow n_{m^*} + 1$;
return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{\hat{M}}$;

Largest-Utilization-First (LUF^+) - for RM Scheduling

Input: $\mathbf{T}$;

1: re-index (sort) tasks such that $\frac{C_i}{T_i} \geq \frac{C_j}{T_j}$ for $i < j$;
2: $\mathbf{T}_1 \leftarrow \emptyset, U_1 \leftarrow 0, n_1 \leftarrow 0; \hat{M} \leftarrow 1$;
3: **for** $i = 1$ to N, where $N = |\mathbf{T}|$ **do**
4: find a processor m^* with $U_{m^*} + \frac{C_i}{T_i} \leq (n_{m^*} + 1)\left(2^{\frac{1}{n_{m^*}+1}} - 1\right)$;
5: **if** no such a processor exists **then**
6: $\hat{M} \leftarrow \hat{M} + 1, \mathbf{T}_{\hat{M}} \leftarrow \emptyset, U_{\hat{M}} \leftarrow 0,\ n_{\hat{M}} \leftarrow 0$;
7: $m^* \leftarrow \hat{M}$;
8: assign task τ_i onto processor m^*, where
$U_{m^*} \leftarrow U_{m^*} + \frac{C_i}{T_i}, \mathbf{T}_{m^*} \leftarrow \mathbf{T}_{m^*} \cup \{\tau_i\}, n_{m^*} \leftarrow n_{m^*} + 1$;
9: return task assignment $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{\hat{M}}$;

Properties

- The time complexity is $O((N + M)\log(N + M))$
- If a solution is derived, the task assignment is feasible by using RM.

A Simple Analysis

- The schedulability test $U_{m^*} + \frac{C_i}{T_i} \leq (n_{m^*} + 1)\left(2^{\frac{1}{n_{m^*}+1}} - 1\right)$ is upper bounded by 69.3%.
- According to the above analysis for EDF, we can also conclude that the utilization is at least $\frac{0.693\hat{M}}{2}$.
- Therefore, the approximation factor of LUF^+ is $\frac{2}{0.693} \approx 2.887$.

A More Precise Analysis

- If $\hat{M}$ is 1, we know that $\sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i} \leq N(2^{\frac{1}{N}} - 1)$.
- Suppose that the processor used by Algorithm LUF^+ is $\hat{M} \geq 2$.
- Let k be the index of the task, at which processor $\hat{M}$ is allocated when running LUF^+. We only look at the iteration when i is k. Therefore,

$$U_k + \sum_{\tau_i \in \mathbf{T}_m} U_i > (n_m + 1)\left(2^{\frac{1}{n_m+1}} - 1\right), \qquad \forall m = 1, \ldots, \hat{M} - 1.$$

- By the sorting of the tasks, we also know that $U_i \geq U_k$ for any $i \leq k$. This also implies that $\sum_{\tau_i \in \mathbf{T}_m} U_i > n_m \left(2^{\frac{1}{n_m+1}} - 1\right)$.
- $x\left(2^{\frac{1}{x+1}} - 1\right)$ is an increasing function of x when $x \geq 1$.
- Let q be the minimum number of tasks assigned on a processor before task τ_k, i.e., $1 \leq q \leq n_m, \forall m = 1, \ldots, \hat{M} - 1$. The approximation factor is $\sqrt{2} + 1$ since

$$U_k + \sum_{i=1}^{k-1} U_i > (1 + (\hat{M} - 1)q)\left(2^{\frac{1}{q+1}} - 1\right) \geq \hat{M}(\sqrt{2} - 1) \approx 0.414\hat{M}.$$

Remarks (Augmenting the Number of Processors)

Survey by Davis and Burns (ACM Computing Surveys, 2011):

Table 3: Approximation Ratios.

Algorithm	Approximation Ratio ($\Re_A$)	Ref.
RMNF	2.67	[Dhall and Liu 1978]
RMFF	2.33	[Oh and Son 1993]
RMBF	2.33	[Oh and Son 1993]
RRM-FF	2	[Oh and Son 1995]
FFDUF	2	[Davari and Dhall 1986]
RMST	$1/(1-u_{max})$	[Burchard et al. 1995]
RMGT	7/4	[Burchard et al. 1995]
RMMatching	3/2	[Rothvoß 2009]
EDF-FF	1.7	[Garey and Johnson 1979]
EDF-BF	1.7	[Garey and Johnson 1979]

Results for Constrained- and Arbitrary-Deadline Systems

	implicit deadlines	constrained deadlines	arbitrary deadlines
partitioned with EDF	$\frac{4}{3} - \frac{1}{3M}$ (Graham 1969) $(1+\epsilon)$ (Hochbaum/Shmoys 1987)	$3 - \frac{1}{M}$ (Baruah/Fisher 2006) $2.6322 - \frac{1}{M}$ (Chen/Chakraborty 2011)	$4 - \frac{2}{M}$ (Baruah/Fisher 2005) $3 - \frac{1}{M}$ (Chen/Chakraborty 2011)
partitioned with DM	(bin-packing) $\frac{7}{4}$ (Burchard et al. 1995) (bin-packing) 1.5 (Rothvoß2009)	$3 - \frac{1}{M}$ (Baker/Fisher/Baruah 2009) 2.84306 (Chen 2016)	$4 - \frac{2}{M}$ (Baker/Fisher/Baruah 2009) $3 - \frac{1}{M}$ (Chen 2016)

The above factors are for speed-up factors, except the two results in partitioned RM scheduling.

Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, Robert I. Davis: On the Pitfalls of Resource Augmentation Factors and Utilization Bounds in Real-Time Scheduling. ECRTS 2017: 9:1-9:25

Outline

Global Scheduling

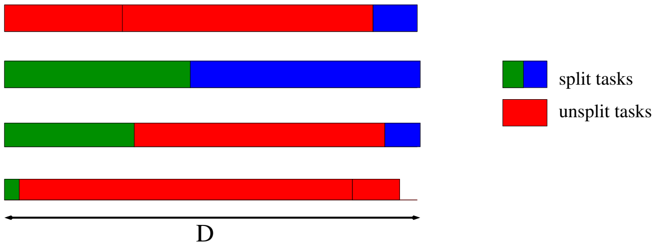
- We will only focus on identical multiprocessors in this module.
- The system has a global queue.
- A job can be migrated to any processor.
- Priority-based global scheduling:
    - Among the jobs in the global queue, the M highest priority jobs are chosen to be executed on M processors.
    - Task migration here is assumed 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-FP, Global-DM, Global-RM: When a job finishes or arrives to the global queue, the M jobs in the queue with the highest priorities (defined by fixed-priority ordering, deadline-monotonic strategy, or rate-monotonic strategy) are chosen to be executed on M processors.
- Pfair scheduling, and the variances (not discussed in this lecture).

Good News for Global Scheduling

- McNaughton's wrap-around rule for $P|pmtn|C_{\max}$ on M processors (historically, task migration is also called task preemption in the literature)
  - Compute $C_{\max}$ as $\max\{\max_{\tau_i \in \mathcal{T}} C_i, \frac{\sum_{\tau_i \in \mathcal{T}} C_i}{M}\}$
    - Assign the tasks according to any order from time 0 to $C_{\max}$
    - If a task's processing exceeds $C_{\max}$, the task is migrated to a new processor from time 0
    - Repeat the assignment of tasks until all the tasks are assigned
  - The resulting schedule minimizes $C_{\max}$

R. McNaughton. Scheduling with deadlines and loss functions. Management Science, 6:1-12, 1959.

McNaughton's Algorithm: Example



Weakness of Partitioned Scheduling

- Restricting a task on a processor reduces the schedulability
- Restricting a task on a processor makes the problem $\mathcal{N}P$-hard
- The $\mathcal{N}P$-completeness does no hold any more if the migration has *no overhead*.
  - Proportionate Fair (pfair) algorithm introduced by Baruah et al. provides an optimal utilization bound for schedulibility
  - A task set with implicit deadlines is schedulable on M identical processors if the total utilization of the task set is no more than M.
  - The idea is to divide the time line into quanta, and execute tasks proportionally in each quanta.
  - It has very high overhead.
  - There are several variances to reduce the overhead.

Sanjoy K. Baruah, N. K. Cohen, C. Greg Plaxton, Donald A. Varvel: Proportionate Progress: A Notion of Fairness in Resource Allocation. Algorithmica 15(6): 600-625 (1996)

Bad News for Global Scheduling

For Global-EDF or Global-RM, the least upper bound for schedulability analysis is at most 1.

Input:

$M+1$ tasks:

- One heavy task τ_k: $D_k = T_k = C_k$
- M light tasks τ_is: $C_i = \epsilon$ and $D_i = T_i = C_k - \epsilon$, in which ϵ is a positive number, very close to 0.

Sudarshan K. Dhall, C. L. Liu, On a Real-Time Scheduling Problem, OPERATIONS RESEARCH Vol. 26, No. 1, January-February 1978, pp. 127-140.

Bad News for Global Scheduling

For Global-EDF or Global-RM, the least upper bound for schedulability analysis is at most 1.

Input:

$M + 1$ tasks:

- One heavy task τ_k: $D_k = T_k = C_k$
- M light tasks τ_is: $C_i = \epsilon$ and $D_i = T_i = C_k - \epsilon$, in which ϵ is a positive number, very close to 0.

Result:

The M light tasks (with higher priority than the heavy task) will be scheduled on M processors. The heavy task misses the deadline even when the utilization is $1 + M\epsilon$.

Sudarshan K. Dhall, C. L. Liu, On a Real-Time Scheduling Problem, OPERATIONS RESEARCH Vol. 26, No. 1, January-February 1978, pp. 127-140.

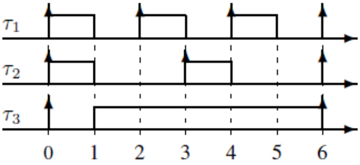
Gold Approach: Resource Augmentation

- The bad news on the least upper bound was very important in 80's, since the research in this direction suffered from the so called "Dhall effect".
- With resource augmentation, by Phillips et al., the "Dhall effect" disappears
  - For Global-EDF, the resource augmentation factor by "speeding up" is $2 - \frac{1}{M}$.
  - That is, if a feasible schedule exists on M processors, applying Global-EDF is also feasible on M processors by speeding up the execution speed with $2 - \frac{1}{M}$.
  - We will focus on schedulability test here first (for the first two parts) and the resource augmentation at the end.

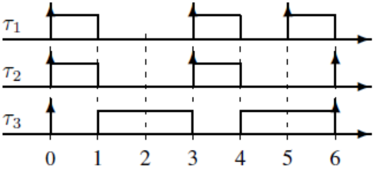
Cynthia A. Phillips, Clifford Stein, Eric Torng, Joel Wein: Optimal Time-Critical Scheduling via Resource Augmentation. STOC 1997: 140-149

Critical Instants?

- The analysis for uniprocessor scheduling is based on the gold critical instant theorem.
- Synchronous release of higher-priority tasks and as early as possible for the following jobs do not lead to the critical instant for global multiprocessor scheduling
  - Suppose that there two identical processors and 3 tasks: (C_i, D_i, T_i) are $\tau_1 = (1, 2, 2), \tau_2 = (1, 3, 3), \tau_3 = (5, 6, 6)$

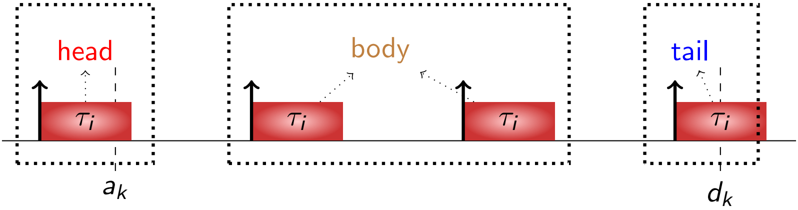


Feasible for τ_3.



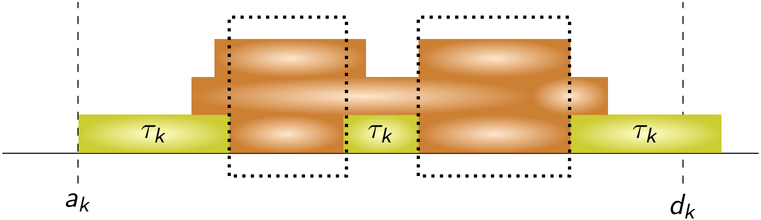
Infeasible for τ_3.

Identifying Interference



- Problem window (interval) is defined in $[a_k, d_k)$.
- The jobs of task τ_i in the problem window can be categorized into three types:
  - Head job (at most one): some computation demand is *carried in* to the problem window for a job arrival before a_k.
  - Body jobs: the computation demand has to be done in the problem window.
  - Tail job (at most one): some computation demand can be *carried out* from the problem window.

Necessary Condition for Deadline Misses



- If τ_k misses the deadline at d_k, there must be at least $D_k - C_k$ units of time in which all M processors are executing other higher-priority jobs.
- Definition: *demand* $W(\Delta)$ in a time interval with length Δ is the total amount of computation that needs to be completed within the interval.
- If τ_k misses its deadline at time d_k, then

$$W(D_k) > M(D_k - C_k) + C_k$$

Summary of Existing Results

Regarding to speedup factors

	implicit deadlines	constrained deadlines	arbitrary deadlines
Global EDF	$2 - \frac{1}{M}$ (Bonifaci et al. 2008)		
Global DM	$3 - \frac{1}{M}$ (Bertogna et al. 2005)	$3 - \frac{1}{M}$ (Baruah et al. 2010)	3 (Chen et al. 2018)
	$\frac{3+\sqrt{7}}{2} \approx 2.823$ (Chen et al. 2015)	3 (Chen et al. 2015)	

Biondi and Sun's Effect?

- The state-of-the-art schedulability analysis have issues for global fixed-priority schedulability and EDF analyses
- For example, if the task set is deemed schedulable under global RM (by using the above schedulability test), there is a *partitioned* schedule which meets all deadlines

- Youcheng Sun, Marco Di Natale: Assessing the pessimism of current multicore global fixed-priority schedulability analysis. SAC 2018: 575-583
- Alessandro Biondi, Youcheng Sun: On the ineffectiveness of 1/m-based interference bounds in the analysis of global EDF and FIFO scheduling. Real-Time Systems 54(3): 515-536 (2018)