Load Regulating Algorithm for Static-Priority Task Scheduling on Multiprocessors

Risat Mahmud Pathan and Jan Jonsson
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden
OUTLINE

- Introduction
- Problem Statement
- Task Model
- Scheduling
  - Feasibility Condition
  - Multiprocessor scheduling
- IBPS Scheduling
- Conclusion
Introduction

- **Real-Time Systems** have timing constraints

- Applications of real-time systems are often modeled as a collection of periodic tasks

- Timing constraints (e.g. deadlines) are stringent in *hard* real-time systems

- **Scheduling** can ensure that all deadlines are met
OUTLINE

• Introduction
• Problem Statement
• Task Model
• Scheduling
  • Feasibility Condition
  • Multiprocessor scheduling
• IBPS Scheduling
• Conclusion
Problem Statement

- **Given**
  - a collection of tasks
  - a collection of available processors

- **the multiprocessor scheduling problem is to determine**
  - whether the tasks can be partitioned among the processors such that all deadlines are met
OUTLINE

• Introduction
• Problem Statement
• **Task Model**
• Scheduling
  • Feasibility Condition
  • Multiprocessor scheduling
• IBPS Scheduling
• Conclusion
Task Model

• Application is modeled as a set of periodic tasks.
  — A task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is to be executed on $m$ processors

• Each task $\tau_i$ has
  — A period $T_i$ (inter-arrival time)
  — A worst-case execution time $C_i$

• Each invocation requires $C_i$ units of execution time before next period
Task Model (cont.)

- Rate-Monotonic (RM) pre-emptive scheduler is used in each processor

- Using RM scheduling, each task $\tau_i$ has a priority. 
  — *The shorter the period, the higher the priority.*

- *Utilization* of a task $\tau_i$ is $u_i = C_i/T_i$

- The *total utilization* of a set $\Gamma$ of tasks is $U(\Gamma) = \sum u_i$
Example Task Set

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>

$u_1 = \frac{5}{10} = 0.5$ \quad $u_2 = \frac{2}{7} = 0.285$ \quad $u_3 = \frac{6}{14} = 0.428$

Total utilization, $U(\Gamma) = u_1 + u_2 + u_3$

$\tau_2$ has the highest priority and $\tau_3$ has the lowest priority.
Rephrased Problem Statement

• How can we guarantee that a set of tasks $\Gamma$ is $RM$ schedulable on $m$ processors?

IBPS

Interval-Based Partitioned Scheduling

The scheduling guarantee using IBPS is given using a feasibility condition.
OUTLINE

• Introduction
• Problem Statement
• Task Model
• Scheduling
  • Feasibility Condition
  • Multiprocessor scheduling
• IBPS Scheduling
• Conclusion
Feasibility Condition

Feasibility Condition of a scheduling algorithm is used to determine (offline) whether all the tasks meet their deadlines during run-time.

- **Necessary and Sufficient, or**
- **Sufficient only**

Necessary and sufficient feasibility test is precise but has higher time complexity.
Sufficient Feasibility Condition

Utilization based sufficient feasibility condition of algorithm $A$ has the following form:

$$A = \text{Uniprocessor Rate-Monotonic(RM) Scheduling (Liu and Layland, 1973)}$$

if $U(\Gamma) \leq n(2^{1/n} - 1)$, then $\Gamma$ is RM-schedulable on uniprocessor.

$$A = \text{Multiprocessor Rate-Monotonic(RM) First-Fit Scheduling (D. Oh 1998)}$$

if $U(\Gamma) \leq 0.41m$, then $\Gamma$ is RM-schedulable on $m$ processors.
If $U(\Gamma) \leq 0.552m$, then $\Gamma$ is IBPS-schedulable on $m$ processors.
OUTLINE

- Introduction
- Problem Statement
- Task Model
- Scheduling
  - Feasibility Condition
  - Multiprocessor Scheduling
- IBPS Scheduling
- Conclusion
Multiprocessor Scheduling

• Two main approaches
  — **Global** (no task assignment, global queue, migration)
  — **Partitioned** (task assignment, local queue, no migration)

• Neither global nor partitioned scheduling can have achievable system utilization more than 50% for static-priority tasks
  (D. Oh et al. 1998, B. Andersson et al. 2001)
Task-Splitting Partitioned Method

• A variation of partitioned scheduling based on task-splitting approach can achieve more than 50%
  – When a task cannot be assigned to a processor, it is split (i.e. migrated during runtime)
  – A bounded number of tasks are migrated
Traditional Partitioned Scheduling

We assume Task 2, Task 1 and Task 3 be the ordering of the tasks to assign to the processors A and B.
Traditional Partitioned Scheduling

Partition Fails!
Task 3 cannot be assigned to any processor because size of Task 3 is too large
**Task-Splitting** Partitioned Scheduling

Different subtasks of Task 3 can be assigned to different processors.

To construct the subtasks, we split Task 3.
**Task-Splitting** Partitioned Scheduling

Different subtasks of Task 3 can be assigned to different processors.

*To construct the subtasks, we split Task 3.*
Task-Splitting Partitioned Scheduling

Partition Success!
OUTLINE

• Introduction
• Problem Statement
• Task Models
• Scheduling
  • Feasibility Condition
  • Multiprocessor Scheduling
• IBPS Scheduling
• Conclusion
IBPS: Basic Idea

• $n$ tasks are grouped in seven utilization subintervals.

• $n$ tasks are assigned to $m$ processor in three phases
  – First two phases has load regulation

• Each processor executes tasks using RM scheduling
IBPS: Basic Idea

• The total utilization in each processor in the first two phases is greater than 55.2% (load regulation)

• All unassigned tasks are assigned in the third phase.

• A task that cannot be assigned to a processor is split.
  — Split a task in exactly two parts, and
  — Each processor only has at most one split task (i.e. \( m/2 \) split tasks)
IBPS: Tasks Grouping in Subintervals

The utilization interval $(0, 1.0]$ is divided into seven utilization subintervals:

- $I_7 = (0, 0.136]$
- $I_6 = (0.136, 0.184]$
- $I_5 = (0.184, 0.221]$
- $I_4 = (0.221, 0.276]$
- $I_3 = (0.276, 0.368]$
- $I_2 = (0.368, 0.552]$
- $I_1 = (0.552, 1.0]$

Each task utilization is within one of the seven utilization subintervals.
IBPS: Seven Utilization Subintervals

Each task is put in the corresponding bucket

- Each subinterval has lower and upper bound
  - For example, $I_2 = (0.368, 0.552)$

- If there are 3 tasks in $I_2$, then the min and max utilization are $(3 \times 0.368)$ and $(3 \times 0.552)$, respectively.
IBPS: Task Assignment

**Phase-1 & Phase-2 Task Assignment Algorithms**

(LOAD REGULATION)

- First two phases assign tasks to $k$ processors such that
  - each of the $k$ processors has load greater than 55.2%
IBPS: Task Assignment

After *Phase-1* and *Phase-2*, the unassigned tasks have special properties.

- These unassigned tasks are called *residue tasks*
  - Total (unassigned) utilization is $U_{\text{res}}$
- For residue tasks, the lower bound $U_{\text{reslow}}$ on $U_{\text{res}}$ is known
  - We have $U_{\text{reslow}} < U_{\text{res}}$

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$I_4$</th>
<th>$I_5$</th>
<th>$I_6$</th>
<th>$I_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>Number of tasks unassigned in $I_2$-$I_6$ are known</td>
<td>$U_7\leq 69%$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IBPS: Task Assignment - Third Phase

Given $U_{reslow} < U_{res}$, how many processors to assign the residue tasks?

\[
0.552 \times x < U_{reslow} \leq 0.552 \times (x+1) \text{ for some } x = 0, 1, 2 \ldots
\]

For example, if $U_{reslow} = 0.01$, then $x = 0$

or, if $U_{reslow} = 0.65$, then $x = 1$

or, if $U_{reslow} = 1.85$, then $x = 3$

$(x + 1)$ processors are used in third phase of task assignment to assign the residue tasks.
Phase-1 & Phase-2 Task Assignment Algorithms

$P_1 \quad L > 55.2\%$

$P_2 \quad L > 55.2\%$

$\ldots$

$P_k \quad L > 55.2\%$

Residue Tasks

Phase-3 Task Assignment Algorithms

$P_{k+1}$

$P_{k+2}$

$\ldots$

$P_{k+x+1}$
IBPS: Feasibility Condition

Theorem: If $U(\Gamma) \leq 0.552m$, then $\Gamma$ is IBPS-schedulable on $m$ processors.

Proof Sketch:

$k$ processors are used in phase 1 and phase 2
$(x+1)$ processors are used in phase 3

We prove that, If $U(\Gamma) \leq 0.552m$, then $(k+x+1) \leq m$. 
Proof Sketch (cont.):

\[ U_{LR} + U_{res} = U(\Gamma) \leq 0.552m \]

if \( k \) processors are used in first two phases, then
\[ 0.552 k < U_{LR} \quad \text{Because of Load Regulation} \]

if at most \( (x+1) \) processors are used in third phase, then
\[ 0.552 x < U_{reslow} < U_{res} \quad \text{Because} \ x0.552 < U_{reslow} \leq (x+1)0.552 \]

Therefore, \( 0.552 k + 0.552 x < U_{LR} + U_{res} \leq 0.552m \)

Or, \( k + x < m \)

Or, \( k + (x+1) \leq m \) \( \text{(Proved)} \)
IBPS and Online Scheduling

If $U(\Gamma) \leq 55.2m$, then all tasks meet deadlines on $m$ processors.

IBPS is applicable for online scheduling

– If $U(\Gamma_{\text{existing}}) + u_{\text{new}} \leq 55.2m$, then task $\tau_{\text{new}}$ is accepted.

– Where to assign the task?
Online Scheduling : O-IBPS

• Load regulation ⇒ third phase requires at most 4 processors (i.e. $x+1 \leq 4$)

• Load regulation enables efficient online scheduling
  — When a task arrives, tasks are reassigned to at most $\min(m, 4)$ processors
  — When a task leaves, tasks are reassigned to at most $\min(m, 5)$ processors

• Therefore, O-IBPS scales very well for large systems.
OUTLINE

• Introduction
• Problem Statement
• Task Models
• Scheduling
  • Feasibility Condition
  • Multiprocessor Scheduling
• IBPS Scheduling
• Conclusion
Conclusion

• *IBPS has many advantages in comparison to other task-splitting algorithms*
  — utilization bound of 55.2%
  — load-regulation
    — online scheduling
    — scalable
  — low overhead of task splitting
    — Only $m/2$ split tasks.
Thank You