Scheduling Multi-User Periodic Arrival of Tasks: Two Linear Programming Formulations

Emmanuel Medernach$^1$ Philippe Lacomme$^1$ Eric Sanlaville$^2$
Claire Hanen$^3$

$^1$medernac@clermont.in2p3.fr, placomme@isima.fr
LPC - IN2P3 and LIMOS, Blaise Pascal University of Clermont-Ferrand

$^2$Eric.Sanlaville@univ-lehavre.fr
LITIS, University of Le Havre

$^3$Claire.Hanen@lip6.fr
LIP6, Université de Paris 10

ASTECE 2009
Outline

1. Problem statement
2. First Linear Program (with transitory phase)
3. Second Linear Program (pattern based)
4. Conclusion
Cyclic scheduling of *periodically occurring* tasks

- A **generic task** is a task which must be performed infinitely often.
- Each occurrence of a generic task is periodically released.

**Known result**

If a schedule exists, then there exists a feasible schedule which is cyclic with a period $T$ equal to the least common multiple of the periods of the individual tasks. [“Scheduling Periodically Occurring Tasks on Multiple Processors” - Lawler, Martel (1980)]

We will suppose that all tasks appear in the first period.
Cyclic scheduling of periodically occurring tasks

Notations

There are $M$ identical machines. A generic task $i$ has
- a task period $\alpha_i$
- a release time $r_i$
- and a required processing time $p_i$.

The schedule period is $T$ (a multiple of all $\alpha_i$). The $j^{th}$ occurrence of task $i$ is denoted by $\langle i, j \rangle$ and has a release date $r_i + (j - 1)\alpha_i$. The number of occurrence of the task $i$ in a period is $T/\alpha_i$.

Condition of feasibility

$$\sum_i \frac{p_i}{\alpha_i} \leq M$$

This condition is necessary. It is also sufficient! (Mc Naughton: preemption at the borders mean several occurrences of the task)
Periodic schedule

Among possible schedules we seek periodic schedules:

**Definition (periodic schedule)**

A schedule is *periodic* if all tasks run periodically (but not necessarily on the same machine each period). The *steady state* begins when all generic tasks have been executed at least once. Time before the steady state is called the *transitory phase*.

- Easy to implement.
- The problem is back to a finite problem.
- But not necessarily dominant (depending on the chosen criterion).
Steady state

The steady state is reached on a period if and only if this period contains the total amount of computation received during each period which is \( \sum_i T \frac{p_i}{\alpha_i} \).

As soon as a period is complete in a periodic schedule it defined a pattern which must repeat itself in all following periods.

Definition (Pattern of a periodic schedule)

The pattern of a periodic schedule is equal to the beginning execution times of all jobs received during a whole period in the steady state.

Note: A periodic schedule defines a pattern but a pattern defines many schedules, because occurrence numbers of a task in a pattern may be shifted without changing the schedule pattern.
### Example of a pattern with 2 machines and a global period of 20

<table>
<thead>
<tr>
<th>$r_i$</th>
<th>$p_i$</th>
<th>$t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>17</td>
</tr>
</tbody>
</table>
Multi-User Problem

- There are many users sending tasks and sharing a computing facility.
- Each task belongs to one user.
- All users could use all available machines.
- **OBJECTIVE**: To give each user a *fair share* of the schedule.
- A *comparison criterion* is used to compare user share of the schedule.
We wish to enforce this following independance property on comparison criterion:

**Independance of users in the comparison criterion**

The comparison criterion used to evaluate a user share of the schedule must depends only on the schedule of this user jobs.
Criterion used: Presence time

Definition (Presence Intervals)

A given user \( U \) is present at time \( t \) if there are tasks of \( U \) waiting or running at time \( t \).

Presence time for a given user is the length of presence intervals for this user in a period after the steady state (because this is the dominant part).

The comparison criterion will depend on the presence times. It reduces to *Flow Time* if each user has only one task.

- **MeanPresenceTime** (or total presence time)
- **MinMaxPresenceTime**
- **WeightedMeanPresenceTime**: We add a weight equal to the total amount of computing of user job. (Stretch)
- **WeightedMinMaxPresenceTime**
Criterion used: Presence time (example)
Pattern: Transitory phase

Dominance
For a given pattern there is only one dominant schedule (modulo tasks locations on machines) for presence interval based criterions.

Given an arbitrary pattern we are able to reach it:

Reachability of a given pattern
It is always possible to reach the steady state after at most $2 + M$ periods or after at most 3 periods if all jobs have length less than the global period.
Problem statement
First Linear Program (with transitory phase)
Second Linear Program (pattern based)
Conclusion

Transitory phase model

- We wish to model a periodic schedule until it finally reaches the steady state.
- The number of time intervals of duration $T$ is fixed to $a$ and we impose that the steady state is reached at the last period $[(a - 1)T, aT]$. 
- If the problem becomes impossible we try again with $a + 1$. We know that it will finally works with $a$ at most $2 + M$. 
- We will use integer programming techniques.
We have $M$ identical machines.

The global schedule period is $T$, which is a multiple of all $\alpha_i$. The number of tasks $i$ released during a period $T$ is $K_i = T/\alpha_i$.

The generic task $i$ belongs to the user $U_i$, has period $\alpha_i$ and a first release date $r_i$ such that the release of task $\langle i, j \rangle$ is $r_{\langle i, j \rangle} = r_i + (j - 1)\alpha_i$.

We have a fixed number of periods $a$ and we impose that the steady state is reached at the last period.

$H$ is a given sufficiently high constant.
Model formulation: Variables

- The execution time of task $\langle i, j \rangle$ is denoted $t_{\langle i, j \rangle}$.
- $m_{i,j,k}$ is a binary variable which is equal to 1 if and only if task $\langle i, j \rangle$ runs on machine $l$, and else it is equal to 0.
- $x_{i_1,j_1,i_2,j_2}$ is a binary variables which is equal to 1 if and only if task $\langle i_1, j_1 \rangle$ runs before task $\langle i_2, j_2 \rangle$ (not necessarily on the same machine), and else it is equal to 0.
Constraints

For all tasks $\langle i, j \rangle$:

- We must respect release times: $t_{\langle i, j \rangle} \geq r_{\langle i, j \rangle}$ and order of tasks: $t_{\langle i, j \rangle} \leq t_{\langle i, j+1 \rangle}$
- All tasks must be allocated to one machine: $\sum_{l=1}^{M} m_{i,j,l} = 1$
- All tasks are periodic: $t_{\langle i, j + K_i \rangle} = t_{\langle i, j \rangle} + T$

For all tasks $\langle i_1, j_1 \rangle \neq \langle i_2, j_2 \rangle$:

- Either a task begins before or after another one:
  $$x_{i_2,j_2,i_1,j_1} + x_{i_1,j_1,i_2,j_2} = 1$$
- $t_{\langle i_2, j_2 \rangle} \leq t_{\langle i_1, j_1 \rangle} + Hx_{i_1,j_1,i_2,j_2}$
- If tasks are allocated on the same machine, then it must not overlap:
  $$\forall l \in \{1 \ldots M\}, \ t_{\langle i_1, j_1 \rangle} + p_{i_1} \leq t_{\langle i_2, j_2 \rangle} + H(3 - x_{i_1,j_1,i_2,j_2} - m_{i_1,j_1,l} - m_{i_2,j_2,l})$$
- Order is the same between global periods:
  $$x_{i_1,j_1,i_2,j_2} = x_{i_1,j_1+K_1,i_2,j_2+K_2}$$
Linearization of Presence Time

We cut tasks to look only inside the last interval \([(a - 1)T, aT]\). For that we apply to all tasks the function

\[ x \mapsto \max\left(\min(x, aT), (a - 1)T\right) \]

This allows to compute the amount of tasks inside the last period, we enforce that this quantity is equal to \(\sum_i T \frac{p_i}{\alpha_i}\).
Linearization of Presence Time

\( \mu \) is a binary variable such that \( \mu = 1 \) if \( y \leq x \) and else \( \mu = 0 \).

\[
\begin{align*}
\mu H + y & \geq x \\
(1 - \mu)H + x & \geq y
\end{align*}
\]

**Minimum:** \( z = \min(x, y) \)

\[
\begin{align*}
z & \leq x \\
z & \leq y \\
(1 - \mu)H + z & \geq y \\
\mu H + z & \geq x
\end{align*}
\]

**Maximum:** \( z = \max(x, y) \)

\[
\begin{align*}
z & \geq x \\
z & \geq y \\
(1 - \mu)H + x & \geq z \\
\mu H + y & \geq z
\end{align*}
\]
We extract for each task of a given user his tasks presence times. Blue part is the waiting interval, Orange part is the execution interval.
Release dates are sorted in non-decreasing order.
Linearization of Presence Time

\[ \sum \min(maxend_i, r_{i+1}) - r_i \]
Linearization of Presence Time

\[ \sum_{i} \min(maxend_i, r_{i+1}) - r_i \]
Linearization of Presence Time

\[ \sum_{i} \min(maxend_i, r_{i+1}) - r_i \]
Linearization of Presence Time

\[ \sum_i \min(maxend_i, r_{i+1}) - r_i \]
Linearization of Presence Time

\[ \sum_{i} \min(maxend_i, r_{i+1}) - r_i \]
Linearization of Presence Time

\[ \sum_i \min(maxend_i, r_{i+1}) - r_i \]
Results

2 users and 4 machines.
User $U_1$: $\alpha_1 = 4$, $p_1 = 10$, $r_1 = 4$
User $U_2$: $\alpha_2 = 4$, $p_2 = 6$, $r_2 = 0$

<table>
<thead>
<tr>
<th>Periods</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinMaxPresence</td>
<td>27s</td>
</tr>
<tr>
<td>WeightedMinMaxPresence</td>
<td>1.7s</td>
</tr>
<tr>
<td>MeanPresence</td>
<td>24s</td>
</tr>
<tr>
<td>WeightedMeanPresence</td>
<td>16s</td>
</tr>
</tbody>
</table>

*Table:* Execution times

WeightedMeanPresence, $a = 5$: 
## Results

3 users and 3 machines

User $U_1$: $\alpha_1 = 3, p_1 = 2, r_1 = 0$

User $U_2$: $\alpha_2 = 6, p_2 = 4, r_2 = 0$

User $U_3$: $\alpha_3 = 6, p_3 = 8, r_3 = 0$

<table>
<thead>
<tr>
<th>Periods</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinMaxPresence</td>
<td>1s</td>
<td>21s</td>
</tr>
<tr>
<td>WeightedMinMaxPresence</td>
<td>5m15</td>
<td>7m43</td>
</tr>
<tr>
<td>MeanPresence</td>
<td>2m10</td>
<td>72m4</td>
</tr>
<tr>
<td>WeightedMeanPresence</td>
<td>28s</td>
<td>4m20</td>
</tr>
</tbody>
</table>

**Table**: Execution times

MeanPresence, $a = 4$: 

![Diagram showing execution times for MeanPresence with $a = 4$.]
Pattern based model

- This second formulation works directly on patterns.
- We model pattern and evaluate its performances for all users.
- If a pattern is found we know how to reconstruct a schedule based on it. We could also use the pattern found inside the first model to compute the best possible transitory phase.
- Inside a period of the steady state, a task may be cut in pieces.
- In the steady state the machines must process all the computing quantity received during a period.
**Components of a pattern**

**Definition (Borderline tasks)**

A task $i$ is a **borderline task** for the pattern if $t_i + p_i \geq T$.

A borderline task is associated with virtual tasks, a **starting task** $[t_i, T]$, an **ending task** $[0, t_i + p_i \mod T]$ and eventually **whole period tasks** $[0, T]$.

**Pattern feasibility**

A pattern is feasible if and only if the inner tasks and corresponding virtual tasks of borderline tasks are schedulable.
Components of a pattern

- Whole period pieces (in black)
- End of task pieces (in blue)
- Start of task pieces (in red)
- Inner tasks (in yellow)
Whole period tasks

A borderline task may be divided in \( n \) or \( n + 1 \) whole period pieces. Let \( \delta_i \) be a binary variable and \( a_{i,j} \) be the number of whole period tasks of the borderline task \( \langle i, j \rangle \). Then:

\[
a_{i,j} + \delta_{i,j} = \left\lfloor \frac{p_{i,j}}{T} \right\rfloor
\]
Variables

$\beta_{i,j}$ is a binary variable equal to 1 if task $\langle i, j \rangle$ is an inner task and equal to 0 if this is a borderline task.

\[
\begin{align*}
    t_{i,j} + p_i & \leq T + (1 - \beta_{i,j})H \\
    \beta_{i,j}H + t_{i,j} + p_i & \geq T
\end{align*}
\]

For each task $\langle i, j \rangle$, $\text{start}_{i,j}$ and $\text{end}_{i,j}$ are the length of the starting and the ending part of a borderline task.

\[
\begin{align*}
    0 & \leq \text{start}_{i,j} \leq T \\
    0 & \leq \text{end}_{i,j} \leq T \\
    p_i(1 - \beta_{i,j}) & = \text{start}_{i,j} + a_{i,j}T + \text{end}_{i,j} \\
    \text{start}_{i,j} & \leq T - t_{i,j} + \beta_{i,j}H \\
    \text{start}_{i,j} & \geq T - t_{i,j} - \beta_{i,j}H
\end{align*}
\]
machine_{i,j,k}^{start}, machine_{i,j,k}^{inner}, machine_{i,j,k}^{end} are binary variables equal to 1 if the starting, inner or ending respectively part of the task \langle i, j \rangle runs on machine k and else equal to 0.
Each piece is at most on one machine:

\forall i, j, \sum_{k=1}^{M} machine_{i,j,k}^{end} = 1 - \beta_i \\
\forall i, j, \sum_{k=1}^{M} machine_{i,j,k}^{start} = 1 - \beta_i \\
\forall i, j, \sum_{k=1}^{M} machine_{i,j,k}^{inner} = \beta_i \\

There is at most one starting or ending piece per machine:

\forall k, \sum_{i,j} machine_{i,j,k}^{end} \leq 1 \\
\forall k, \sum_{i,j} machine_{i,j,k}^{start} \leq 1
Available machines

We sort the machines so that the whole period tasks are always on the first machines. This means that machines from 1 to $\sum_{i,j} a_{i,j}$ are used by whole period tasks. It remains $M - \sum_{i,j} a_{i,j}$ machines. Let $\gamma_k$ be a binary variable such that $\gamma_k = 0$ if the machine $k$ is used by whole period tasks and else $\gamma_k = 1$.

$$\forall k, \gamma_{k+1} \geq M = \sum_{i,j} a_{i,j} + \sum_k \gamma_k$$

As we know that $a_{i,j} \geq \lfloor p_i / T \rfloor - 1$, we already have:

$$\forall k \in \{1, \ldots, \sum_{i,j} \lfloor p_i / T \rfloor - 1\}, \gamma_k = 0$$
No overlapping of tasks

\[ \forall (i_1, j_1) \neq (i_2, j_2), \]

\[
x_{i_1,j_1, i_2,j_2} + x_{i_2,j_2, i_1,j_1} = 1
\]

\[
t_{i_2,j_2} \leq t_{i_1,j_1} + H x_{i_1,j_1, i_2,j_2}
\]

\[
k, t_{i_1,j_1} + p_{i_1} \leq t_{i_2,j_2} + H (3 - x_{i_1,j_1, i_2,j_2} - mach_{i_1,j_1,k}^{interieur} - mach_{i_2,j_2,k}^{interieur})
\]

On a given machine all inner and starting (resp. ending and inner) tasks must begin after the end (resp. must end before the starting) task present on that machine if it exists.

\[ \forall i_1, j_1, i_2, j_2, i_3, j_3, k, \]

\[
end_{i_1,j_1} \leq t_{i_2,j_2} + (2 - machine_{i_1,j_1,k}^{end} - machine_{i_2,j_2,k}^{inner})H
\]

\[
t_{i_2,j_2} + p_{i_2} + start_{i_3,j_3} \leq T + (2 - machine_{i_3,j_3,k}^{start} - machine_{i_2,j_2,k}^{inner})H
\]

\[
end_{i_1,j_1} + start_{i_3,j_3} \leq T + (2 - machine_{i_3,j_3,k}^{start} - machine_{i_1,j_1,k}^{end})H
\]
Schedule reconstruction

From the pattern we have to build a periodic schedule. We have to shift tasks to the next period if it does not respect its release:
Let $t'_{i,j}$ be the real running time, if $t_{i,j} \geq r_{i,j}$ alors $t'_{i,j} = t_{i,j}$, else $t'_{i,j} = t_{i,j} + T$. Thus:

$$
t'_{i,j} = t_{i,j} + \theta_{i,j}T, \theta_{i,j} \in 0, 1
$$

$$
t'_{i,j} \geq r_{i,j}
$$

Presence times are computed from this reconstructed schedule.
Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Jobs</td>
<td>2</td>
<td>4</td>
<td>13</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Machines</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>MinMaxPresence</td>
<td>&lt;0.01s</td>
<td>0.01s</td>
<td>0.22s</td>
<td>0.06s</td>
<td>0.17s</td>
</tr>
<tr>
<td>WeightedMinMaxPresence</td>
<td>&lt;0.01s</td>
<td>0.01s</td>
<td>0.51s</td>
<td>1.03s</td>
<td>4.51s</td>
</tr>
<tr>
<td>MeanPresence</td>
<td>&lt;0.01s</td>
<td>0.01s</td>
<td>0.45s</td>
<td>0.37s</td>
<td>0.11s</td>
</tr>
<tr>
<td>WeightedMeanPresence</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.51s</td>
<td>0.56s</td>
<td>0.19s</td>
</tr>
</tbody>
</table>

Problem 5, WeightedMinMaxPresence:
Summary and Conclusion.

- We defined criterions based on the presence interval of each user.
- We have proposed 2 linear programming models for the scheduling of multi-user periodic arrival of tasks.
  - Transitory model
  - Pattern-based model: much smaller, much faster
- Current work: accurate tests of second formulation

- even second formulation will probably not be able to solve medium to large size instances.
- We are now working on heuristics for that problem, based on the Vehicule Routing Problem