Pre Run-time Scheduling – Flexibility Integration Offline - Online

Gerhard Fohler
Mälardalen University, Sweden
gerhard.fohler@mdh.se
Offline schedules

- general timing constraints
- offline scheduler
  - resolves constraints
  - constructs one solution which meets all constraints
- fixed (blind) runtime execution
- no flexibility

- how can we
  - increase flexibility
  - add dynamic tasks
  - integrate with online scheduling methods
The diagram illustrates a state transition in a real-time system. The nodes and states are labeled as follows:

- **N0**: Starts with node A, transitions to node B.
- **N1**: Transitions from node C to node D.

The diagram also shows the dependency relation `dl(PG)`, indicating a dependency between states or processes.
Slot Shifting

Offline

• timing constraints ✓
• offline schedule ✓
• we have
  – offline constructed schedule

• we want
  – include dynamic tasks
  – schedule them online

• what can we do?
  – include in offline schedule (e.g., pseudo periodic) inefficient
  – fit into empty slots no guarantees
  – we can do better!
Basic Idea

shift A’s slots

NW

CPU

aper

c=1

shift A’s slots
Shifting pre-runtime tasks

• pre runtime schedule assigns fixed times for execution
  – although different times possible
  – overconstrains schedule
    • we have to select one out of several possible times
    • ...for the sake of algorithm only

• we know, that we can shift A
  – execute the aperiodic task at once
  – feasibility of tasks not violated
  – how much and where can we shift?
  – what are boundaries?
Limitations on Shifting

we can shift tasks

limitations

• receipt of message
• sending of message
• earliest start time of precedence graph, end-to-end constraints, task chain
• deadline of -

calculate start time, deadline pairs for tasks

• expresses flexibility of task
• reduces overconstraining
• fit in aperiodic task by shifting as long as these constraints met
\[dl(PG)\]

\[\text{est} \quad \text{dl} \quad \text{block (AB)}\]

\[\text{NW} \quad \text{est} \quad \text{block (CD)}\]

\[\text{est} \quad \text{dl} \quad \text{N0} \quad \text{N1}\]
These tasks are assigned fixed starttimes or deadlines.
(Subgraphs of precedence graphs allocated to nodes combined.)

independent tasks with starttimes, deadlines on single nodes

simple EDF runtime scheduling
Slot Shifting …

Offline

- timing constraints
- offline schedule
- earliest start times, deadlines ✓
How much shifting?

• know what is
  – earliest time to start task
  – latest time to finish

• aperiodic arrives: how far can we shift static task?
- **latest start time**: start no later or violate deadline.
- **have to ensure when executing aperiodics**
  - how?
- **more complex dispatching**: still next task, but check for constraints.
- **more memory**: 3 integers per task.
Slot Shifting

Offline
- timing constraints
- offline schedule
- earliest start times, deadlines
- latest start times ✓
so far soft aperiodics

how can we decide?

• need idle resources for aperiodics
• before deadline of aperiodic
• which resources can we use?
Spare Capacities

- spare capacities, \( sc = \) length of execution interval
  - execution times
- available for aperiodic tasks
- know amount and location from schedule!

\[ \text{unused resources, spare capacities} \]
sc = 5 - 3 = 2

est | dl
N0

block (AB)

sc = 6 - 0 = 6

est | block (CD) | dl
NW

sc = 8 - 3 = 5

A | B | C | D

est
N1
Slot Shifting

Offline

- timing constraints
- offline schedule
- earliest start times, deadlines
- latest start times
- spare capacities
  - not yet...
Intervals

sort deadlines  *disjoint intervals:*

• *end*: deadline of task(s)
• tasks with that deadline
• *spare capacity, sc*  
  the amount of idle resources in that *interval*
• *start*: max of est of task(s) and end previous interval

• empty intervals:
  – $\text{end}(I_{i-1}) < \text{start}(I_i)$
  – $\text{wcet} = 0$
\[ \text{sc}(I_{(AB)}) = 5 - 3 = 2 \]

\[ \text{sc}(I_{(X)}) = 3 - 3 = 0 \]

\[ wcet(T) \]

..almost the truth…
• intervals \hspace{0.5cm} execution intervals!
\[ \text{sc}(I_{(AB)}) = 5 - 3 = 2 \]

\[ \text{sc}(I_{(X)}) = 3 - 4 = -1 \]

\[ \text{sc}(I_{(AB)}) = 5 - 3 - 1 = 1 \]

“borrowing”
borrowing mechanism:

- if tasks in subsequent interval need more resources than available in it: execute in other interval, use resources from there “borrow”
- run-time mechanisms resolve negative spare capacity
- only for calculation and flexibility
- start of interval can be earliest start time
- earliest start time checked separately
Slot Shifting

Offline
• timing constraints
• offline schedule
• earliest start times, deadlines
• latest start times
• intervals ✓
• spare capacities ✓
Online Mechanisms- Scheduling

online scheduler invoked at each node after each slot

- check for new aperiodic tasks
- guarantee algorithm
- take scheduling decision
- update spare capacities
- execute scheduling decision

earliest deadline first
• after each slot, scheduling decision taken locally at each node
  – no ready task:
    CPU idle
  – \( sc(I_c) > 0 \), soft aperiodic task A:
    execute A
  – \( sc(I_c) = 0 \):
    an offline or guaranteed task has to be executed or
deadlines are missed
  takes care that no latest start time is missed!
  no other mechanism needed, eg, watchdog, etc
  implicit invocation, no extra memory needed
  – \( sc(I_c) > 0 \), soft aperiodic task:
    offline or guaranteed task executed
Acceptance of aperiodics

- aperiodics (without deadline): $sc > 0$: one slot can be given to it
- firm aperiodics (wcet and deadline): want them executed either completely or not at all

\textit{guarantee algorithm} $O(N)$
- aperiodic task A \((r, wcet, dl)\)
- three parts of spare capacities available

- \(\text{sc}(I_c)\): remaining sc in current interval
- \(\text{sc}(I_i)\): \(\text{sc}(I_i) > 0, \ c < i \ \ \ \ \text{end}(I_i) \ \ \ \text{dl}(A), \ \text{end}(I_{i+1}) > \text{dl}(A)\), sc in all \textit{full} intervals between \(r\) and \(dl\)
- \(\min(\text{sc}(I_{i+1}), \ \text{dl}(A) - \text{dl}(I))\), minimum spare capacities of last interval or up to the deadline of aperiodic in last interval
- possibly interval split
Guarantee

• if sum of total sc between dl and r are larger or equal wcet, guarantee
• need to ensure guarantees resources are not used otherwise
• after guarantee:
  – update interval 1
  – update interval 1-1
  – ...
  – update interval c
Spare capacities at runtime

- aperiodic execution
  - decrease spare capacity of current interval

\[ I_{(AB)}, \ \text{sc}(I_{(AB)}) = 2 \]

\[ I_{(X)}, \ \text{sc}(I_{(X)}) = 0 \]

at \( t: \ \text{sc}(I_{(AB)}) = 2 - 1 = 1 \)

\[ I_{(X)}, \ \text{sc}(I_{(X)}) = 0 \]
• no execution
  – decrease spare capacity of current interval

\begin{align*}
  &I_{(AB)} \text{, } \text{sc}(I_{(AB)})=2 \\
  &I_{(X)} \text{, } \text{sc}(I_{(X)})=0
\end{align*}

at \( t \): \( \text{sc}(I_{(AB)})=2-1=1 \)\[N_0\]
• execution of offline task $T$
  - $T$ current interval $I_c$
spare capacity stays the same

at $t$: $\text{sc}(I_{(AB)})=2$

$\text{sc}(I_{(AB)})=2$ $\text{sc}(I_{(X)})=0$

$\text{sc}(I_{(X)})=0$

$I_{(X)}$
• execution of offline task $T$
  – $T$ future interval $I_f$
    • spare capacity $I_c$ decreased
    • spare capacity $I_f$ increased

\begin{align*}
  l_{(AB)} \quad sc(l_{(AB)})=2 & \quad l_{(X)} \quad sc(l_{(X)})=0 \\
  \text{at } t: sc(l_{(AB)}) &= 2-1=1 \\
  \text{at } t: sc(l_{(X)}) &= 0+1=1
\end{align*}
• update capacity of $I_f$
  – if $0 \ldots$ done
  – if $< 0 \ldots$ need to update previous interval $I_{f-1}$

• $sc(I_{f-1})$
  – if $0 \ldots$ done
  – if $< 0 \ldots$ need to update previous interval $I_{f-2}$

• ….

• until $sc \ 0$ or $I_c$
Shifting Messages

• communication medium resource like CPU from scheduling perspective
• shift messages as well
• restriction to sending messages earlier
  – no receiver synchronization necessary
  – may increase spare capacities at receiver
  – when message received - spare capacities updated
  – else same
Analysis

- MARS
- 4 CPUs
- TDMA network
- ~1600 task sets generated and pre runtime scheduled
- randomly generated aperiodic tasks
- each point in plots 700-1000 task sets
- 0.95 confidence intervals < 5%
local shifting

combine load

guarantee ratio

dl = 2 * wcet

dl = wcet
The diagram shows the relationship between the combined load and the guarantee ratio for two scenarios: $dl = 2 \times wcet$ and $dl = wcet$. As the combined load increases, the guarantee ratio decreases for both scenarios, with the $dl = 2 \times wcet$ scenario consistently maintaining a higher guarantee ratio compared to the $dl = wcet$ scenario.
“Slot shifting nouveau”

- further acceptance test
- integration with TBS
- ...

<table>
<thead>
<tr>
<th></th>
<th>Periodic with constraints</th>
<th>Sporadic</th>
<th>Aperiodic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple</td>
<td>Complex</td>
<td>Firm</td>
</tr>
<tr>
<td>Offline</td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Online</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Sch</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

- Periods
- Deadlines
- Start times

- End-to-end dl
- Inst. separation
- Distribution
- Jitter etc.

- Minimum separation between instances

- Deadlines
- Guarantee

- No dl
Slot Shifting - Summary

- handle online tasks while maintaining feasibility of offline scheduled tasks
- offline reduction of complexity
- simple runtime handling
- “interface” for integration of offline and online scheduling
- offline scheduled system for critical activities
- restrict amount of shifting
- flexibility for rest

predictable flexibility
Articles

• Gerhard Fohler
  *Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems*

• Damir Isovic, Gerhard Fohler
  *Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints*
Novel Applications

mix of activities and demands
• **core system** with high demands
  – strict timing behavior
  – safety critical, fault tolerant
  – proven and tested for worst case
• **hard real-time applications**
  – temporal correctness, etc.
• **flexible real-time applications**
  – not completely known
  – some deadlines can be missed
• **non real-time activities**
  – must not disturb real-time activities
TT: high cost even for non critical applications

ET: not deterministic behavior of critical activities
offline, TT
original temporal constraints

offline scheduler

flexibility analysis

scheduling table

target windows of tasks

online, ET

EDF tasks

EDF scheduling

FPS tasks

FPS scheduling

reuse of scheduling components

offline scheduling

EDF tasks

FPS tasks
target windows control flexibility of task execution

- target window = original task execution
  - no flexibility, original schedule
- target window after flexibility analysis
  - flexibility of execution while meeting demands
- reduced target windows
  - reduced flexibility, e.g., for jitter control
- modifying target windows selects flexibility of tasks individually
Meeting Novel Application Demands

• core system
  offline scheduling

• hard real-time applications
  offline scheduling or online scheduling

• flexible real-time applications
  combined offline/online approach

• non real-time activities
  together with combined offline/online

• flexibility individually configured

• guaranteed tasks protected
Predictably flexible real-time systems

- Predictably flexible real-time systems
- Hard real-time appl.
- Core system
deterministic

TT

flexible RT appl.

non RT appl.

ET