Adaptive Scheduling of Streams in Real-time Applications

Marek Balej, Zdenek Hanzalek
Czech Technical University in Prague
Contents

• Time-triggered Approach and Problem description
• Scheduling with time constraints
• Adaptive scheduling, i.e. adding new messages/nodes into the existing schedule
Search space of real-time problems

Complexity of the optimal synthesis problem is comparable the one of the verification/analysis problem.

In practice: sub-optimal solution, found in a part of the search tree (e.g. by heuristic algorithm) has practical value, but partial verification (i.e. the one which does not consider all possible behaviors) has none.

It is easier to synthesize the time-constrained system than to leave the freedom to the designer and consequently verify its time properties.
Problem Description

• Time-triggered scheduling
• Non-preemptive scheduling
• Routing is given (e.g. tree topology)
• Centralized algorithm
• Respecting time constraints
(Non-adaptive) Scheduling

Basic assumptions
- Tree topology
- switch integrated in each node (special HW)
- full duplex

Time-triggered interval
- highest-priority
- strictly isochronous - Precision Transparent Clock Protocol
- data are forwarded according to a static communication schedule
Input Parameters of the Scheduling Problem

List of links
- link delay

<table>
<thead>
<tr>
<th>link</th>
<th>$N_1 \to N_3$</th>
<th>$N_1 \to N_4$</th>
<th>$N_1 \to N_2$</th>
<th>$N_2 \to N_1$</th>
<th>$N_3 \to N_1$</th>
<th>$N_4 \to N_1$</th>
<th>$N_3 \to N_5$</th>
<th>$N_5 \to N_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{LD}$ [ns]</td>
<td>4875</td>
<td>5130</td>
<td>5862</td>
<td>3841</td>
<td>4875</td>
<td>4895</td>
<td>4875</td>
<td>4875</td>
</tr>
</tbody>
</table>

\[ T_{LD} = T_{TxD} + T_{CD} + T_{RxD} + T_{ad} + T_{BD} \]

List of messages
- source
- destination(s)
- transmission delay
- required
  - release date
  - deadline
  - end-to-end delay
- multicast message – used e.g. for synchronization

<table>
<thead>
<tr>
<th>ID</th>
<th>path</th>
<th>$T_{TD}$ [ns]</th>
<th>$\tilde{r}$ [ns]</th>
<th>$\tilde{d}$ [ns]</th>
<th>$\tilde{e}$ [ns]</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>$N_2 \to N_3$</td>
<td>5760</td>
<td>5000</td>
<td>20000</td>
<td>11000</td>
</tr>
<tr>
<td>257</td>
<td>$N_3 \to N_2$</td>
<td>5760</td>
<td>15000</td>
<td>40000</td>
<td>15000</td>
</tr>
<tr>
<td>258</td>
<td>$N_1 \to N_3$</td>
<td>5760</td>
<td>15000</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>259</td>
<td>$N_3 \to N_1$</td>
<td>5760</td>
<td>20000</td>
<td>35000</td>
<td>–</td>
</tr>
<tr>
<td>128</td>
<td>$N_3 \to {N_1, N_2, N_4, N_5}$</td>
<td>11680</td>
<td>5000</td>
<td>{-,,-,18000}</td>
<td>{-,17675,17675,15000}</td>
</tr>
</tbody>
</table>
Problem Refinement

The objective is to find the shortest schedule for the TT interval based on a network topology/parameters, message parameters and required position in the schedule

- messages on the same link are separated with a minimum inter-message gap (added to the transmission delay $T_{TD}$)
- as soon as the first bit of a message is received, it may be forwarded to another link, i.e. if $T_{LD} < T_{TD}$, two nodes may process a different part of the same message at the same time

overlapping precedence relation
Solution of TT scheduling

Formulation in terms of the Resource Constrained Project Scheduling with Temporal Constraints minimizing the schedule makespan (denoted $PS|\text{temp}|C_{\max}$)

Tree topology of nodes
• determines the rooting of messages

Unicast message
• chain of tasks executed on dedicated communication links
• chain starts at the source node and ends at the destination node

Multicast message
• tree of tasks
Modeling by PS\temp\C_{\text{max}}

Task execution corresponds to a transmission of a message on the respective link

- **Transmission delay** - processing time equal to $T_{TD}$
- **Link delay** - edge with positive weight $T_{LD}$
- **Release date** – edge with positive weight from dummy task to source task
- **Deadline** - edge with negative weight from sink task to dummy task
- **Required end-to-end delay** - edge with negative weight from sink task to source task
ILP formulation of the problem

input: vector $p$, processing time of the tasks
vector $a$, assignment of tasks to links
adjacency matrix $W$

internal variables: $x_{ij}$ is equal to 1 iff task $T_i$ is followed by task $T_j$

output: vector $s$, start time of the tasks
makespan $C_{max}$

\[
\begin{align*}
\min \ C_{max} \\
\text{subject to:} \\
& s_j - s_i \geq w_{ij}, \quad \forall i, j \in 1, \ldots, n | w_{ij} > -\infty \\
& s_i - s_j + UB \cdot x_{ij} \geq p_j, \quad \forall i, j \in 1, \ldots, n | i > j \land a_i = a_j \\
& s_j - s_i + UB \cdot (1 - x_{ij}) \geq p_i, \quad \forall i, j \in 1, \ldots, n | i > j \land a_i = a_j \\
& s_i + p_i \leq C_{max}, \quad \forall i \in 1, \ldots, n \\
\end{align*}
\]

where: $x_{ij} \in \{0, 1\}$; $s_i, C_{max} \in \mathbb{R}_0^+$ and $UB$ is a constant such that $UB > C_{max}$
Scheduling

• Results returned by a heuristic using MTS (Most Total Successors) priority rule for the same instance

<table>
<thead>
<tr>
<th>Topology</th>
<th>n</th>
<th>Cmax (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>47,20</td>
</tr>
<tr>
<td>2</td>
<td>290</td>
<td>116,00</td>
</tr>
<tr>
<td>3</td>
<td>440</td>
<td>150,40</td>
</tr>
<tr>
<td>4</td>
<td>960</td>
<td>219,20</td>
</tr>
<tr>
<td>5</td>
<td>510</td>
<td>120,26</td>
</tr>
<tr>
<td>6</td>
<td>880</td>
<td>154,66</td>
</tr>
</tbody>
</table>

• Schedules are computed in a fraction of time compared to ILP
• Cmax is close to optimal one, computation times are in the range of 10ms
Adaptive Scheduling  
(also called Rescheduling)

- What if we need to add a new message/node into the existing schedule?

- **Free rescheduling** - make a new schedule from scratch – finds a new place for all tasks (both new and original ones) - the algorithm is the same

- **Fixed rescheduling** - add new tasks without moving the original ones

- Schedule may “degenerate” (increase of makespan of the schedule) if we fix positions of original tasks, but scheduling process is faster (it schedules only new tasks)

- Degeneration is not very big, depends on the topology and may be eliminated by free rescheduling from time to time
Schedule with new messages and with fixed original tasks (fixed rescheduling)

- Another instance for the same topology
- Blue tasks were in the original schedule
- The new message contains new tasks 10 and 11
- Task 11 caused the prolongation (it must wait until execution of his predecessor which cannot be placed earlier in the schedule – this breaks the priority rule)
Completely new schedule for the same messages (free rescheduling)

- The same instance as on the previous slide
- If we make a new schedule from all tasks (both original and new), task 10 will be placed earlier because it has higher priority than task 9 and its successor can be executed earlier in the schedule and in this case, the resulting schedule is optimal
Effect of adding new message

• If we cannot move the original tasks, we cannot use the priority rule for the whole schedule (it works only with messages added in current time)

• Adding a task with high priority often causes a prolongation of the schedule

• This problem occurs mainly on the critical resource

• The length of prolongation is dependent on the number of nodes traversed (number of tasks) starting from the critical resource

• The prolongations do not sum-up, i.e. rescheduling can make use of the unused space in the previous schedule
Experiments

- Tests of degeneration of schedule makespans (a comparison of the two previous ways – free and fixed rescheduling)
- Results are an average of **300 instances**
- Messages have different transmission delays - 6880, 4000, 2560 ns
- Messages were added in cycles. **5 messages were added in each cycle.**
- This schedule was compared with the schedule where all the same tasks were scheduled at once.
- Messages were generated at random
- Using two algorithms for fixed rescheduling:
  - **fixed 1** – we cannot move the tasks that were already scheduled
  - **fixed 2** – we can move the tasks that were already scheduled if they have no successors (this change cannot expand into the whole schedule)
- Tested for 5 priority rules, but there are no big differences between them (the results shown in this presentation apply to MST priority rule)
- Tested for two schemes but parallel scheme is more complex in rescheduling than the serial scheme
Parameters of the tests

Inputs
• **Topology** – there are 7 different topologies
• **Cycles** – number of cycles for adding new messages
• **n** – number of tasks in the original schedule
• **n_{new}** – number of added tasks (sum for all cycles)

Outputs
• \( C_{max}^{free} \) – average makespan of schedule when all tasks are scheduled at once
• \( C_{max}^{fixed} \) – average makespan of schedule when tasks are added in cycles
• \( \Delta C_{max} \) – percentage difference between two previous makespans
• \( \Delta C_{max}^{worst} \) – the biggest difference in makespans from all instances (percentage)
• \( T_{C}^{free} \) – average time of scheduling of all tasks at once (this is not directly comparable with the next parameter, because if we use scheduling of all tasks, we must do it in every cycle but \( T_{C}^{free} \) contains only time for scheduling after the last cycle)
• \( T_{C}^{fixed} \) – sum of scheduling all scheduling cycles
• \( d^{free} \) – average end to end delay for the first way
• \( d^{fixed} \) – same as the previous for fixed original tasks
Results – small instances (10 cycles)

<table>
<thead>
<tr>
<th>Topology</th>
<th>n</th>
<th>$n_{new}$</th>
<th>$C_{\text{max}}^{\text{free}}$ (µs)</th>
<th>$T_C^{\text{free}}$ (ms)</th>
<th>$d^{\text{free}}$ (µs)</th>
<th>$C_{\text{max}}^{\text{fixed}}$ (µs)</th>
<th>$\Delta C_{\text{max}}$ (%)</th>
<th>$\Delta C_{\text{max}}^{\text{worst}}$ (%)</th>
<th>$T_C^{\text{fixed}}$ (ms)</th>
<th>$d^{\text{fixed}}$ (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>228</td>
<td>244</td>
<td>114,888</td>
<td>5</td>
<td>37,835</td>
<td>128,631</td>
<td>11</td>
<td>26</td>
<td>0</td>
<td>34,744</td>
</tr>
<tr>
<td>2</td>
<td>340</td>
<td>373</td>
<td>129,928</td>
<td>11</td>
<td>51,821</td>
<td>163,419</td>
<td>25</td>
<td>53</td>
<td>1</td>
<td>46,688</td>
</tr>
<tr>
<td>3</td>
<td>347</td>
<td>370</td>
<td>129,736</td>
<td>11</td>
<td>49,944</td>
<td>164,21</td>
<td>26</td>
<td>48</td>
<td>0</td>
<td>46,599</td>
</tr>
<tr>
<td>4</td>
<td>599</td>
<td>526</td>
<td>164,457</td>
<td>24</td>
<td>63,71</td>
<td>224,802</td>
<td>36</td>
<td>58</td>
<td>2</td>
<td>57,56</td>
</tr>
<tr>
<td>5</td>
<td>615</td>
<td>526</td>
<td>163,96</td>
<td>25</td>
<td>70,206</td>
<td>226,122</td>
<td>37</td>
<td>61</td>
<td>1</td>
<td>64,192</td>
</tr>
<tr>
<td>6</td>
<td>1186</td>
<td>697</td>
<td>196,689</td>
<td>70</td>
<td>88,536</td>
<td>268,092</td>
<td>36</td>
<td>52</td>
<td>6</td>
<td>84,784</td>
</tr>
<tr>
<td>7</td>
<td>416</td>
<td>217</td>
<td>163,955</td>
<td>4</td>
<td>59,263</td>
<td>169,318</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>56,804</td>
</tr>
</tbody>
</table>

- For small instances $\Delta C_{\text{max}}$ is big in some cases
- Big differences in $\Delta C_{\text{max}}$ and $\Delta C_{\text{max}}^{\text{worst}}$ between difference topologies
- Fixed rescheduling is much faster
- No big differences between fixed rescheduling 1 and 2
## Results – medium instances (25 cycles)

<table>
<thead>
<tr>
<th>Topology</th>
<th>n</th>
<th>( n_{\text{new}} )</th>
<th>( C_{\text{max}}^{\text{free}} ) (μs)</th>
<th>( T_{\text{C}}^{\text{free}} ) (ms)</th>
<th>( d_{\text{free}}^{\text{free}} ) (μs)</th>
<th>( C_{\text{max}}^{\text{fixed}} ) (μs)</th>
<th>( \Delta C_{\text{max}} ) (%)</th>
<th>( \Delta C_{\text{worst}}^{\text{max}} ) (%)</th>
<th>( T_{\text{C}}^{\text{fixed}} ) (ms)</th>
<th>( d_{\text{fixed}}^{\text{fixed}} ) (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>228</td>
<td>607</td>
<td>178,918</td>
<td>12</td>
<td>51,299</td>
<td>193,666</td>
<td>8</td>
<td>17</td>
<td>2</td>
<td>44,816</td>
</tr>
<tr>
<td>2</td>
<td>340</td>
<td>927</td>
<td>214,227</td>
<td>28</td>
<td>72,499</td>
<td>247,833</td>
<td>15</td>
<td>30</td>
<td>4</td>
<td>58,058</td>
</tr>
<tr>
<td>3</td>
<td>347</td>
<td>933</td>
<td>216,036</td>
<td>29</td>
<td>70,355</td>
<td>250,534</td>
<td>15</td>
<td>30</td>
<td>6</td>
<td>57,207</td>
</tr>
<tr>
<td>4</td>
<td>599</td>
<td>1313</td>
<td>235,024</td>
<td>59</td>
<td>87,282</td>
<td>303,912</td>
<td>29</td>
<td>46</td>
<td>8</td>
<td>69,46</td>
</tr>
<tr>
<td>5</td>
<td>615</td>
<td>1312</td>
<td>244,262</td>
<td>60</td>
<td>92,665</td>
<td>307,484</td>
<td>25</td>
<td>44</td>
<td>10</td>
<td>75,912</td>
</tr>
<tr>
<td>6</td>
<td>1186</td>
<td>1749</td>
<td>267,356</td>
<td>142</td>
<td>111,457</td>
<td>349,302</td>
<td>30</td>
<td>55</td>
<td>22</td>
<td>97,321</td>
</tr>
<tr>
<td>7</td>
<td>416</td>
<td>545</td>
<td>242,189</td>
<td>9</td>
<td>83,153</td>
<td>249,904</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>76,546</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topology</th>
<th>( C_{\text{max}}^{\text{fixed}} ) (μs)</th>
<th>( \Delta C_{\text{max}} ) (%)</th>
<th>( \Delta C_{\text{worst}}^{\text{max}} ) (%)</th>
<th>( T_{\text{C}}^{\text{fixed}} ) (ms)</th>
<th>( d_{\text{fixed}}^{\text{fixed}} ) (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>184,234</td>
<td>3</td>
<td>14</td>
<td>0</td>
<td>46,61</td>
</tr>
<tr>
<td>2</td>
<td>241,61</td>
<td>12</td>
<td>31</td>
<td>2</td>
<td>60,149</td>
</tr>
<tr>
<td>3</td>
<td>244,542</td>
<td>13</td>
<td>31</td>
<td>2</td>
<td>58,974</td>
</tr>
<tr>
<td>4</td>
<td>301,311</td>
<td>26</td>
<td>44</td>
<td>7</td>
<td>72,219</td>
</tr>
<tr>
<td>5</td>
<td>304,813</td>
<td>24</td>
<td>38</td>
<td>7</td>
<td>78,06</td>
</tr>
<tr>
<td>6</td>
<td>345,445</td>
<td>30</td>
<td>49</td>
<td>21</td>
<td>97,875</td>
</tr>
<tr>
<td>7</td>
<td>243,139</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>77,329</td>
</tr>
</tbody>
</table>

- The percentage \( \Delta C_{\text{max}} \) is lower than in smaller instances and the absolute value of \( \Delta C_{\text{max}} \) is very similar (no increase)
Results – big instances (100 cycles)

- Still no increase in absolute value of $\Delta C_{\text{max}}$, so the percentage values are again lower.
- For big instances, values of $d_{\text{fixed}}$ are much better than values of $d_{\text{free}}$.
- Now we can see that the second algorithm for fixed scheduling shows a little better results for $\Delta C_{\text{max}}$, but the computation time is longer.
We are looking for collaboration

- API related to application your experience
  - End-to-end dealy
  - Absolute time windows (deadlines, release dates)
  - Time windows relative to some event
  - Synchronization of tasks
  - Parameters for adaptivity
    - fixed messages-nodes .... combination of free and fixed resched.
    - constraints to be respected in mode changes
  - Redundance issues
- Typical topologies/data for benchmarks
- Integration of our algorithms to your products