Providing Weakly-Hard Guarantees using TWCA

Sophie Quinton – INRIA Grenoble Rhône-Alpes

Beyond the Deadline

ESWeek Tutorial, October 15, 2017

Outline

Basics of TWCA

Improvements using combinations

How to obtain input data for TWCA through tracing

Extensions of TWCA

Conclusion and perspectives

Outline

What is TWCA?

TWCA:

- means **Typical Worst-Case Analysis**
- is a method for computing **weakly-hard** bounds on response times and deadline misses.
- applies to systems with **sporadic overload**
- does not provides guarantees for sporadic tasks

What is TWCA?

TWCA:

- means **Typical Worst-Case Analysis**
- is a method for computing **weakly-hard** bounds on response times and deadline misses.
- applies to systems with **sporadic overload**
- does not provides guarantees for sporadic tasks

Advantages

- This approach is computationally efficient
- m-out-of-k constraints are easy to understand
- We make no assumptions w.r.t. dependencies

TWCA in a nutshell

Principle:

- Identify typical bounds for the behavior of a system and how often the system may leave these bounds

TWCA in a nutshell

Principle:

- Identify typical bounds for the behavior of a system and how often the system may leave these bounds

Output for each task: a set of weakly-hard guarantees

TWCA in a nutshell

Principle:

- Identify typical bounds for the behavior of a system and how often the system may leave these bounds

Output for each task: a set of weakly-hard guarantees

- Response-time view:
  1. a hard bound on its response times: *WCRT*
  2. a so-called typical bound: *TWCRT*
  3. a function *err* s.t. out of every k consecutive executions, at most $err(k)$ response times may be larger than *TWCRT*

TWCA in a nutshell

Principle:

- Identify typical bounds for the behavior of a system and how often the system may leave these bounds

Output for each task: a set of weakly-hard guarantees

- Response-time view:
  1. a hard bound on its response times: *WCRT*
  2. a so-called typical bound: *TWCRT*
  3. a function *err* s.t. out of every k consecutive executions, at most $err(k)$ response times may be larger than *TWCRT*
- Deadline miss view: a **deadline miss model**, i.e., a function *dmm* such that out of every k consecutive executions, at most $dmm(k)$ jobs may miss their deadline.

Basic principle: preliminary definitions

System model

- Uniprocessor with Fixed-Priority Preemptive (FPP) scheduling
- Tasks: C_i, π_i and an **activation model**

Activation model: we use arrival curves

- $\delta_i^-(k)$ lower bounds the minimum size of an interval containing k activations of task i
- δ_i^- can be converted into a time-based functions η_i^+
- non-sporadic tasks also have an upper bound δ_i^+

Basic principle: preliminary definitions

System model

- Uniprocessor with Fixed-Priority Preemptive (FPP) scheduling
- Tasks: C_i, π_i and an **activation model**

Activation model: we use arrival curves

- $\delta_i^-(k)$ lower bounds the minimum size of an interval containing k activations of task i
- δ_i^- can be converted into a time-based functions η_i^+
- non-sporadic tasks also have an upper bound δ_i^+

For TWCA, tasks have 3 curves:

- a worst-case bound δ_i^-
- a typical bound $\delta_{i,typ}^-$
- an overload bound $\delta_{i,over}^-$

Basic principle: preliminary definitions

Level-i **quiet time**: instant t such that all tasks of priority higher than or equal to i released strictly before t have completed at t.

Level-i **busy window**: interval $[t_1, t_2[$ such that:

- a task with a priority higher than or equal to i is activated at t_1;
- t_1 and t_2 are level-i quiet times;
- there is no other level-i quiet time between t_1 and t_2.

Basic principle: preliminary definitions

Level-i **quiet time**: instant t such that all tasks of priority higher than or equal to i released strictly before t have completed at t.

Level-i **busy window**: interval $[t_1, t_2[$ such that:

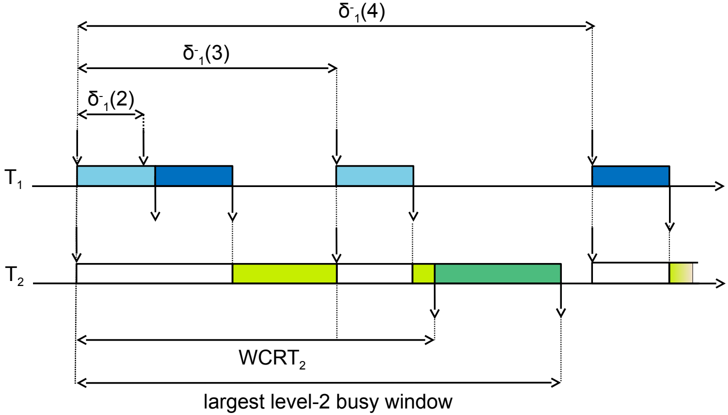
- a task with a priority higher than or equal to i is activated at t_1;
- t_1 and t_2 are level-i quiet times;
- there is no other level-i quiet time between t_1 and t_2.

The longest level-i busy window is bounded by $BW_i = B_i^+(K_i)$ where

$$B_i^+(q) = C_i \times q + \sum_{j \in hpe(i)} (\eta_j^+(B_i^+(q)) \times C_j)$$

$$K_i = min\{q \geq 1 \mid B_i^+(q) \leq \delta_i^-(q+1)\}$$

Basic principle: preliminary definitions



Basic principle: computation of *WCRT* and *TWCRT*

- The worst-case response time of task i is bounded by

$$WCRT_i = \max_{1 \leq q \leq K_i} \{B_i^+(q) - \delta_i^-(q)\}$$

- $TWCRT_i$ is obtained following the same approach but using the $\delta^-_{i,typ}$ curves.

Basic principle: computation of *WCRT* and *TWCRT*

- The worst-case response time of task i is bounded by

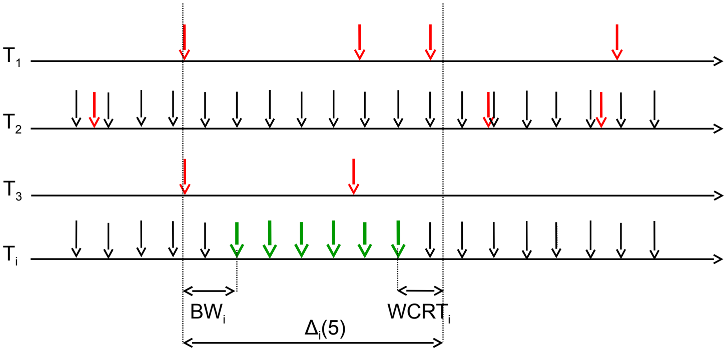
$$WCRT_i = \max_{1 \leq q \leq K_i} \{B_i^+(q) - \delta_i^-(q)\}$$

- $TWCRT_i$ is obtained following the same approach but using the $\delta_{i,typ}^-$ curves.

We now focus on the computation of err_i: the number of jobs in a sequence of k consecutive executions that may have a response time larger than *TWCRT*

Basic principle: computation of err_i

1. compute $\Delta_i(k)$, the time interval during which a higher priority overload activation may impact one of the k activations
2. bound the number of overload activations of each higher priority task in $\Delta_i(k)$
3. bound their impact



Basic principle: computation of err_i

1. $\Delta_i(k) = BW_i + \delta_i^-(k) + WCRT_i$
2. number of overload activations of each higher priority task in $\Delta_i(k)$ is bounded by $\eta_i^+(\Delta_i(k))$
3. impact of each overload activation: at most K_i

Basic principle: computation of err_i

1. $\Delta_i(k) = BW_i + \delta_i^-(k) + WCRT_i$
2. number of overload activations of each higher priority task in $\Delta_i(k)$ is bounded by $\eta_i^+(\Delta_i(k))$
3. impact of each overload activation: at most K_i

$$err_i(k) = K_i \times \sum_{j \in hpe(i)} \eta_{j,over}^+ \Delta_i(k)$$

Basic principle: computation of err_i

1. $\Delta_i(k) = BW_i + \delta_i^-(k) + WCRT_i$
2. number of overload activations of each higher priority task in $\Delta_i(k)$ is bounded by $\eta_i^+(\Delta_i(k))$
3. impact of each overload activation: at most K_i

$$err_i(k) = K_i \times \sum_{j \in hpe(i)} \eta^+_{j,over} \Delta_i(k)$$

The impact of each activation is largely overestimated!
Example: not all activations in the worst-case busy window miss their deadlines ($N_i \leq K_i$).

Outline

Basics of TWCA

Improvements using combinations

How to obtain input data for TWCA through tracing

Extensions of TWCA

Conclusion and perspectives

Improvements using combinations

NB: Focus on deadline misses rather than response times

Schedulable combination $\bar{c}$: a set of tasks that may experience overload in the same busy window without any deadline miss

Improvements using combinations

NB: Focus on deadline misses rather than response times

Schedulable combination $\bar{c}$: a set of tasks that may experience overload in the same busy window without any deadline miss

Improved deadline miss model:

$$dmm_i(k) = \min_{\bar{c} \in \mathcal{S}} \{dmm_i^{\bar{c}}(k)\}$$

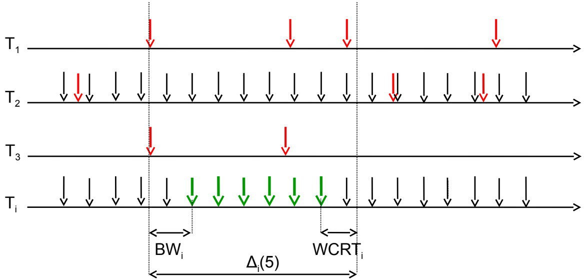
where

$$dmm_i^{\bar{c}}(k) = N_i \times \sum_{\substack{j \in hpe(i) \\ j \notin \bar{c}}} \eta^{+}_{j,over}(\Delta_i(k))\}$$

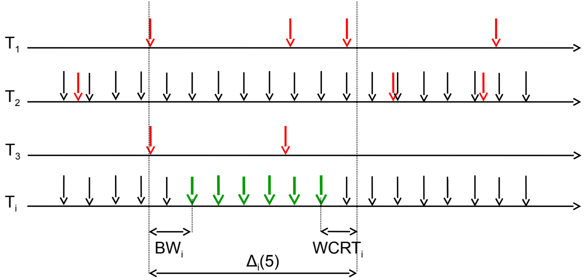
and $\mathcal{S}$ denotes the set of schedulable combinations ($\mathcal{U}$ is the set of schedulable combinations).

Improvements using combinations

Further improvement: knapsack problem formulation where the objective is to pack as many unschedulable combinations as possible into $\Delta_i(k)$



Improvements using combinations

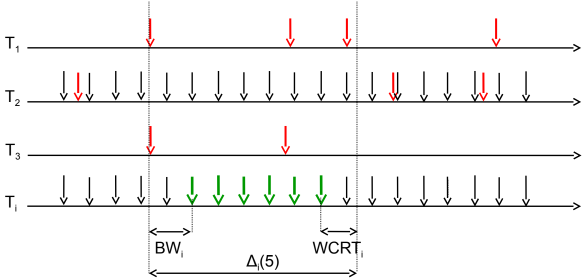


Improved deadline miss model:

$$dmm_i(k) = \max\{N_i \times \sum_{\bar{c} \in \mathcal{U}} x_{\bar{c}} \mid \forall j \in hpe(i), \sum_{\substack{\bar{c} \in \mathcal{U} \\ \text{s.t. } j \in \bar{c}}} x_{\bar{c}} \leq \eta^{+}_{j,over}(\Delta_i(k))\}$$

where $x_{\bar{c}}$ is the number of busy windows which correspond to $\bar{c}$

Improvements using combinations



Improved deadline miss model:

$$dmm_i(k) = \max\{N_i \times \sum_{\bar{c} \in \mathcal{U}} x_{\bar{c}} \mid \forall j \in hpe(i), \sum_{\substack{\bar{c} \in \mathcal{U} \\ \text{s.t. } j \in \bar{c}}} x_{\bar{c}} \leq \eta^{+}_{j,over}(\Delta_i(k))\}$$

where $x_{\bar{c}}$ is the number of busy windows which correspond to $\bar{c}$
$\longrightarrow$ ILP problem

Outline

Using traces to get the input models

Trace analysis and overload extraction

- based on assumptions similar to derived worst-case analysis
- automated overload extraction possible for some activation models: e.g. mixed messages in a CAN bus

Outline

Extensions of TWCA

- extension to FPNP (other policies in progress)
- TWCA at the runnable level
- TWCA for task chains
- TWCA for budgeting (TAS case study)
- TWCA in presence of limited buffers

Outline

Conclusion and perspectives

Summary: TWCA so far

- uniprocessor
- static priority (non) preemptive scheduling
- dependent tasks with arbitrary activation patterns

Case studies

- Anonymized trace from an OEM
- CAN bus analysis for Daimler
- TAS case study

Work in progress

- extension to multiprocessor systems
- identification of the main sources of pessimism in the analysis