FROM RESEARCH TO INDUSTRY

cea tech

# Maximizing the execution rate of low-criticality tasks in mixed-criticality system

Mathieu Jan, Lilia Zaourar
CEA LIST – LaSTRE
&
Maurice Pitel
Schneider Electric Industries

www.cea.fr

leti & list

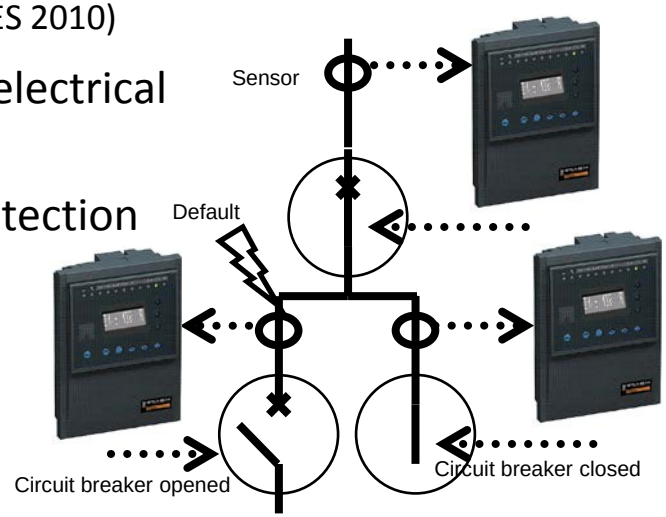
# Motivation for this work

- Feedback from Schneider Electric on our Time-Triggered based RTOS
  - Sizing must be made using the Worst-Case Execution Time (WCET) of each task and high margins are taken
  - Very low probability to simultaneously have the WCET for each task
  - Huge over-sizing of the CPU resources compared to what is needed in average

- True …
  - … but worst-case situation is required by certification authorities for hard real-time systems
  - … and economical constraints push for the use of these unused resources for the execution of low-criticality tasks

# Use case & problem statement

- Medium voltage protection relays (SIES 2010)
  - Safety-function: detect and isolate faults in the electrical network
  - End-to-end temporal constraint between the detection of power faults and asking the tripping of circuit breakers
  - SIL2 certification level IEC 61508




- Embed less (or non) safety functionalities
  - Display information, optimizing the distribution of energy, etc.
  - Different levels of criticality: Mixed-Criticality (MC) systems
    - We are only interested in the use of two levels of criticality
- Enable the design of MC systems where
  - Taken separately high and low-criticality tasks are schedule but the union is not
  - Slack should be used to optimize the execution rate of low-criticality tasks
    - Should not be simply dropped

# Related work and task model

- Well-known Vestal task model for MC systems
  - But unused slack time for executing low-criticality tasks which are dropped
- Elastic MC task model
  - Low-criticality tasks have a desired period, a max period and a set of possible early-release points
  - But how tasks can be « compressed » is based on their utilization
- For the low-criticality tasks, extend the periodic task model with …
  - Stretching period factors: deadline is a flexible parameter
    - Set or range of possible (bounded) values specified off-line
    - Applied when a deadline is going to be missed, in order to postpone it
  - Importance level: which low-criticality task should be stretched first
  - Sub-problems we consider
    - Schedulability analysis extended for maximizing the use of CPU
    - On-line algorithm to set stretching factor values, in particular in a Time-Triggered paradigm

# Notations

- A set of $n$ independent synchronous, preemptible and implicit-deadline periodic tasks: $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$
  - $n_{high}$ high-criticality task ( $\Gamma_{ct}$ ) with a utilization noted $U_{high}$
  - $n_{low}$ low-criticality tasks ( $\Gamma_{nct}$ ) with a utilization noted $U_{low}$
  - Temporal parameters of a task $\tau_i$ : $(P_i, C_i, D_i)$
- Low-criticality tasks have additional parameters
  - Importance level: $V_i$
    - The higher the value, the higher is the importance of the task …
  - Maximum stretching factor that can be applied: $S_{i,max}$
    - Defines low utilization bound that can be reached
    - At run-time, the actual value is noted: $S_i$ and $1 \leq S_i \leq S_{i,max}$
- Total utilization noted $U$ and $m$ is the number of processors

# Schedulability analysis

- Computation for each low-criticality task of the minimum required stretching factor $(S_{i,min})$
  - Which worst-case temporal behavior will be used on-line
  - Assuming each task uses it WCET and by definition $S_{i,min} \leq S_{i,max}$

- Constraints
  - On the utilization that can generate the low-criticality tasks due to the presence of the high-criticality task: $U_r = m - U_{high}$

$$U_{low} \leq U_r \Leftrightarrow \sum_{i \in \Gamma_{nct}} \frac{C_i}{S_i \times P_i} \leq U_r$$

  - Bounds on the utilization value of a low-criticality task: $u_{i_{min}} \leq \frac{C_i}{S_i \times P_i} \leq u_{i_{max}}$

- Objective
  - Maximize the utilization of the resources, while stretching the less important low-criticality tasks first

$$Max \sum_{i \in \Gamma_{nct}} V_i \times \frac{C_i}{S_i \times P_i}$$

# On-line decision algorithm

- When it is called?
  - At the beginning of an overloaded situation
  - Within an overloaded situation for other low-criticality tasks
- When it is called, the most important low-criticality task is being executed
  - Hierarchical scheduling within the low-criticality tasks (Alternative: use EDF-VD)

**Algorithm 1** Decision algorithm for setting the stretching factors of low-criticality tasks.

**Require:** $\tau_i \in \Gamma_{nct_k}$ and the current time $t$

1: $S_i \leftarrow ComputeStretching(\tau_i, t, D_i, S_i)$;
2: **if** $S_i \geq S_{i,min}$ **then** Stop $\tau_i$ and log the error; **end if**
3: $D_i \leftarrow S_i * P_i$;
4: UpdateReady($\tau_i$);
5: Call the scheduler;

- When a low-criticality task finishes, its stretching factor is reset to 1

# Using stretching factors within the TT paradigm

- In the Time-Triggered paradigm, the hypothesis of independent task
  - Can be made at the system level but not at the application level
  - Visibility date of data: deadline of the producer
    - To achieve determinism execution behavior
    - A task may only use data whose visibility dates are equals or inferior to its release date
  - The use of stretching factors change the visibility date
    - Inconsistent with the statically defined triggering points

- Gather low-criticality tasks within groups
  - That must be kept temporally consistent between them
  - Use stretching factor and importance level parameters at the group level $\Gamma_{nct_k}$
  - Modification to our linear program: consider the utilization of each group

$$\frac{1}{S_k} \times \sum_{\tau_i \in \Gamma_{nct_k}} \frac{C_i}{P_i}$$

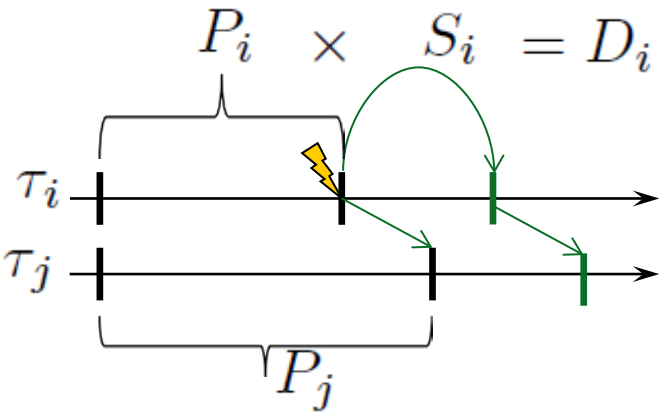
# Decision algorithm within the TT paradigm

**Algorithm 2** Additionnal steps in the decision algorithm when integrated in the TT paradigm, compared to algorithm 1.

**Require:** $\Gamma_{nct_k}$ with $\tau_i \in \Gamma_{nct_k}$

1: **for all** $\tau_j \in \Gamma_k \neq \tau_i$ **do**
2: **if** $\tau_j$ is ready **then** RemoveFromReady($\tau_j$);
3: **else** RemoveFromSleeping($\tau_j$); **end if**
4: **if** $S_i \geq S_{j,min}$ **then** Stop $\Gamma_{nct_k}$, log the error; **end if**
5: $D_j \leftarrow P_j + (D_i - P_i)$;
6: **if** $\tau_j$ is finished **then** SetFlag(Stretched); **end if**
7: InsertReady($\tau_j$);
8: **end for**

- Two constraints
  - Change the visibility date of already produced data
    - But not yet visible, therefore no data inconsistency is possible
  - Maintain the initial offsets between the triggering points



# Preliminary evaluations

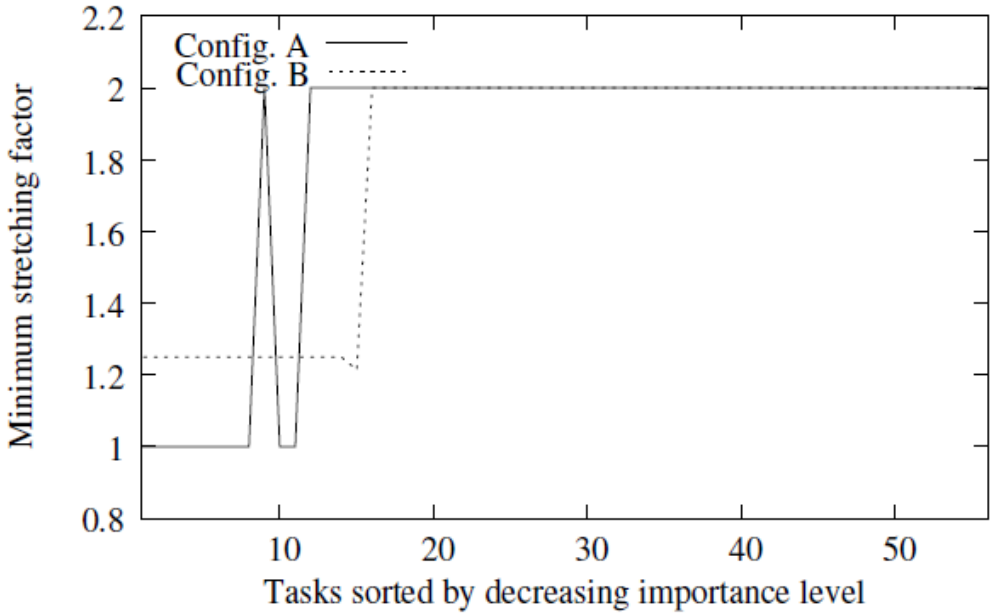
- Task set generator
  - Random task set, utilization computed using UUniFast-Discard algorithm
  - Range of possible periods: 10 to 100 ms
  - Each task is either a high or a low-criticality task until $U_{high}$ reaches 50%
  - $S_{i,max} = 2$
- 3 tasks sets are generated with 20% of high-criticality tasks
  - From 50 – 70 tasks, with 5 – 14 high-criticality tasks
  - Initial utilization set to 125% and 150% off a 2 processors system
- 3 metrics used for the evaluation
  - Average stretching factor for all the low-criticality tasks: $Aver$
  - Average stretching factor for the 25% most important low-criticality tasks: $Aver25+$
  - Average stretching factor for the 75% less important low-criticality tasks: $Aver75$

# Obtained stretching factors

| $U$ | *Aver* | *Aver25+* | *Aver75* | *Aver* w/o $V_i$ |
|---|---|---|---|---|
| 125 | 1.69/1.36/1.59 | 1/1/1 | 1.94/1.48/1.79 | 1.65/1.3/1.48 |
| 150 | 1.86/1.65/1.83 | 1.5/1/1.37 | 2/1.87/2 | 1.97/1.67/1.74 |

- Stretching factors
  - Are reduced for the most important low-criticality tasks
  - Much higher for the less important low-criticality tasks
- Without the importance level parameter
  - Low-criticality must be stretched more when the importance level is used, but can lead to almost unused stretching factors for important low-criticality tasks
- Distribution of stretching factors for two configurations
  - Config. A: random values for the importance level
  - Config. B: 25% of the most important tasks should have

$$S_{i,min} = 1.25$$



# Conclusion and future work

- Proposal of a task model and associated one-line decision algorithm to maximize the execution rate of the low-criticality task
  - Inspired by the elastic task model and opposite approach to ER-EDF
  - Off-line CPU maximization by computing minimum required stretching factors
  - Algorithm to deal with stretching factors within the Time-Triggered paradigm

- Future work
  - Further evaluations: overhead of the different possible strategies for setting the stretching factors
  - Different approach for the execution part through the use of a generalized form of the Time-Triggered approach (eXternal-Triggered)
  - Apply this approach to lessen the deadline miss ratio of the low-criticality task when setting a trade-off with energy consumption
  - Our RTOS partner is evaluating the development of a prototype