FROM RESEARCH TO INDUSTRY

cea tech

Real-Time Scheduling for Time-Triggered Mixed Criticality systems

Mathieu Jan

CEA LIST

www.cea.fr

leti & list

Outline of the talk

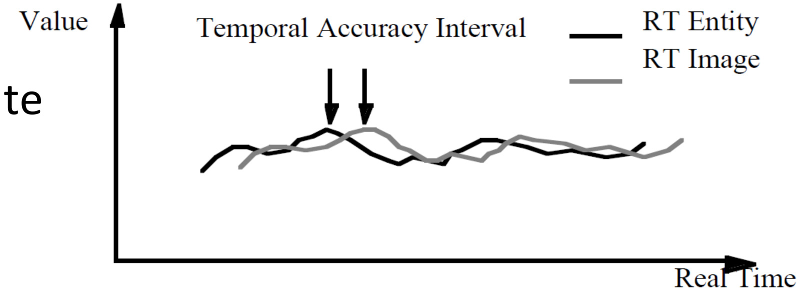
- Real-Time scheduling context
  - The Time-Triggered execution model
  - Mixed Criticality systems: two main task models
- Supporting MC within TT using the Vestal task model
  - Building scheduling tables: two solutions based on Linear Programming
- Supporting MC within TT using the elastic task model
  - Maintaining task scheduling consistency: on-line decision algorithm
- Conclusion and future work

Motivation for this work

- The Time-Triggered (TT) paradigm
  - Used in industrial fields to build hard real-time systems subject to certification constraints
  - Tasks are triggered by the advancement of time
- Certification requirements: temporal behavior is mastered
  - Schedulability must be demonstrated in the worst-case situation
    - Difficulties to compute Worst-Case Execution Time (WCET)
    - Very low probability to simultaneously have the WCET for each task
  - Huge over-sizing of the CPU resources compared to what is needed
- Economical constraints
  - Push for the use of these unused resources
  - A solution: Mixed-Criticality (MC) within TT
    - Unused processing capabilities: for the low-criticality tasks

Time-Triggered paradigm

- The Time-Triggered (TT) paradigm (as introduced by Kopetz)
  - Temporal accuracy of real-time data/entity
    - <value, date>
    - Real-Time Image: is valid if it is an accurate representation in the time and value domains of a real-time entity

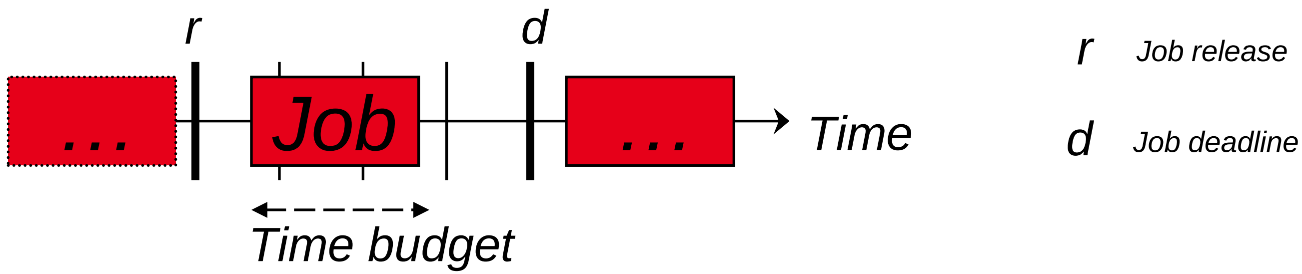


  - Firewalls used at predefined points to exchange RT images
    - Define the minimum validity time of a RT image

- Several flavors
  - When I/O are performed differ as well as they assumed durations
    - Logical Execution Time (LET)
    - Bounded Execution Time (BET)

Our Time-Triggered execution model

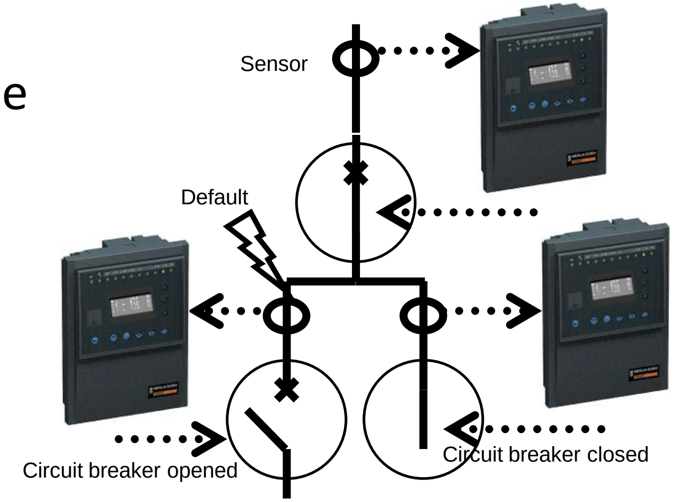
- A Bounded Execution Time flavor
  - Both computations and I/O can be performed whenever between the predefined points
  - A task is a cyclic sequence of jobs with timing constraints
  - By default, the visibility date of a real-time image is equal to the job deadline



- Strict observation principle
  - A job works on real-time image whose visibility dates are inferior or equal to the job release

MC scheduling problem

- Medium voltage protection relays
  - 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
    - Easily demonstrated using the TT paradigm



- Embed additional 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 schedulable

The popular Vestal MC task model

- Rationale
  - Higher the criticality level is, greater the estimated WCET value is
- Periodic task model with
  - Two estimated Worst-Case Execution Time (WCET): C_i (LO), C_i (HI)
  - For the HI-criticality tasks: C_i (LO) < C_i (HI)
  - For the LO-criticality tasks: C_i (LO) = C_i (HI)
  - A criticality level χ_i : LO or HI
- System states
  - Two execution modes: LO and HI
  - Switch to the HI-mode when a HI-criticality task exceeds its C_i (LO)
  - Only the schedulability of HI-criticality tasks ensured while in HI-mode

The elastic task model for MC

- When using the Vestal task model, low-criticality tasks are simply dropped in HI mode
  - Wasting processing power

- For the low-criticality tasks, extend the periodic task model
  - Flavor of the elastic task model
  - Stretching 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

Common notations

- We consider the use of these two tasks models within the TT paradigm

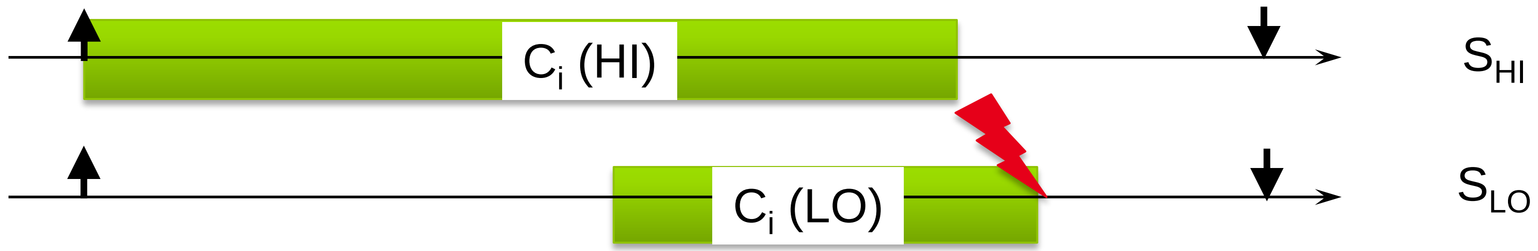
- A set of n independent synchronous, preemptible and implicit-deadline periodic tasks: $\Gamma = \{ \tau_1, \tau_2, \ldots \tau_n \}$
  - Job set of all jobs: J_Γ
  - Temporal parameters (at least) of a task : $\tau_i = (P_i)$
  - Total utilization noted U and m is the number of processors

Outline of the talk

- Real-Time scheduling context
  - The Time-Triggered execution model
  - Mixed Criticality systems: two main task models
- Supporting MC within TT using the Vestal task model
  - Building scheduling tables: two solutions based on Linear Programming
- Supporting MC within TT using the elastic task model
  - Maintaining task scheduling consistency: on-line decision algorithm
- Conclusion and future work

Supporting Vestal MC task model within TT

- Definition of two schedules tables: S_{LO} and S_{HI}
- Main issue: guarantee that a mode change cannot lead to an unfeasible schedule for the HI-criticality tasks
  - Switching occur at specific points: where the HI-criticality tasks can first exceed their C_i (LO) values
  - Remaining time is sufficient to schedule all HI-criticality tasks



- Building S_{HI} and S_{LO} can not be made independently

Approach and related work

- TT MC scheduling using Linear Programming techniques
  - Two proposed solutions: LPMC-HI and LPMC-Both
- LPMC-HI: two separated but linked linear programs
  - First LP: guarantee the schedulability of HI-criticality tasks and maximize the number of completed LO-criticality tasks
  - Second LP: guarantee the schedulability of LO-criticality tasks
  - Differs from [Baruah & Fohler, RTSS 1991]: HI-criticality tasks can be delayed to complete a LO-criticality task
- LPMC-Both: simultaneous building of S_{LO} and S_{HI}
  - Similar to [Theis et al., WMC 2013]
  - HI-criticality tasks are splitted into two sub-jobs: J_i^{LO} and J_i^{Δ}
  - J_i^{Δ} represents the additional WCET assumed when in the HI mode

Specific notations

- Full temporal parameters of a task: $\tau_i = (\chi_i, P_i, C_i(LO), C_i(HI))$
- Hyper-period H is divided in intervals
  - An interval being delimited by two job releases
  - Size of interval k: $|I_k|$, set of jobs in interval k: J_k
  - $W_{j,k}$: weight of job j on interval k (not an execution time but a fraction of it)
- Goal: compute $w_{j,k}^{LO}$ and $w_{j,k}^{HI}$

A scheduling example

- Task set running on a dual-core: m = 2 with H = 12

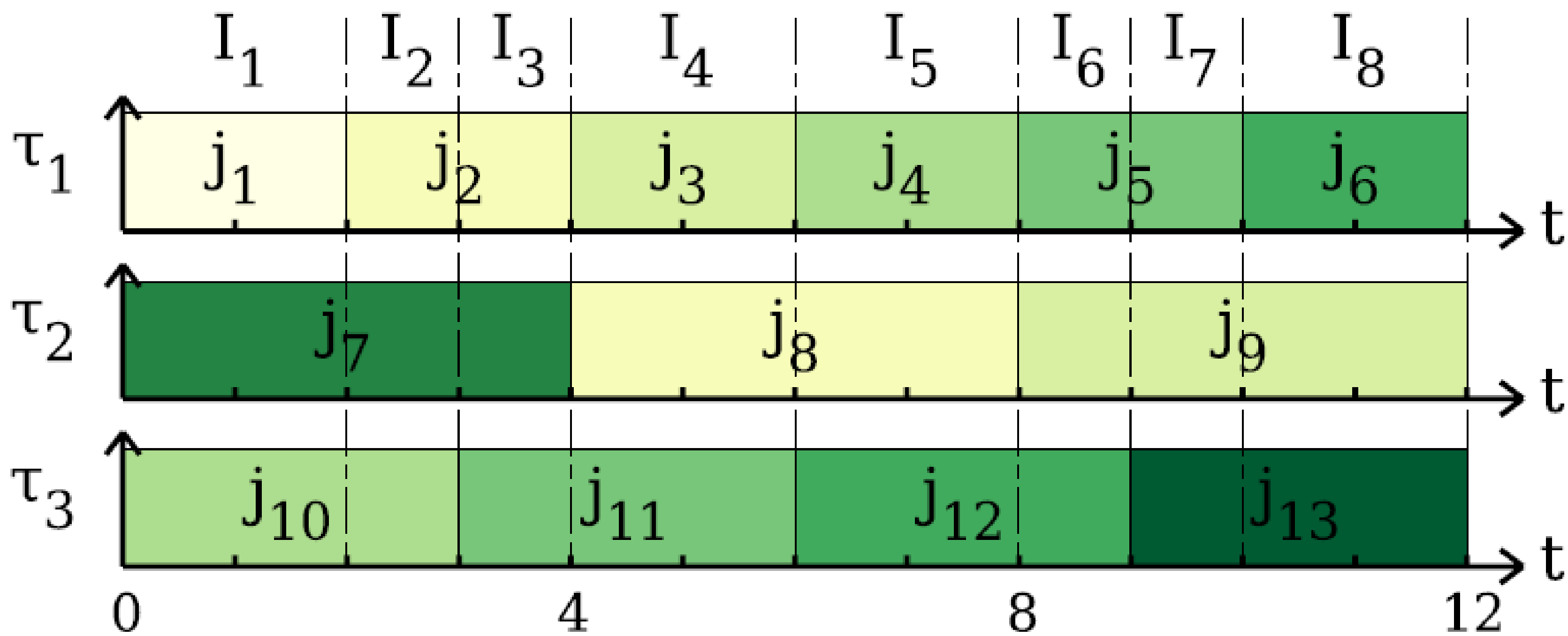
	χ_i	P_i	$C_i(LO)$	$C_i(HI)$
τ_1	LO	2	1.5	1.5
τ_2	HI	4	2	3
τ_3	HI	3	1	2

$\tau_1 \leftarrow J_{LO}$; $\tau_2, \tau_3 \rbrace\ J_{HI}$

U = 2.16

$U_{HI} = 1.41$

$U_{LO} = 1.58$



A scheduling example

- Task set running on a dual-core: m = 2 with H = 12

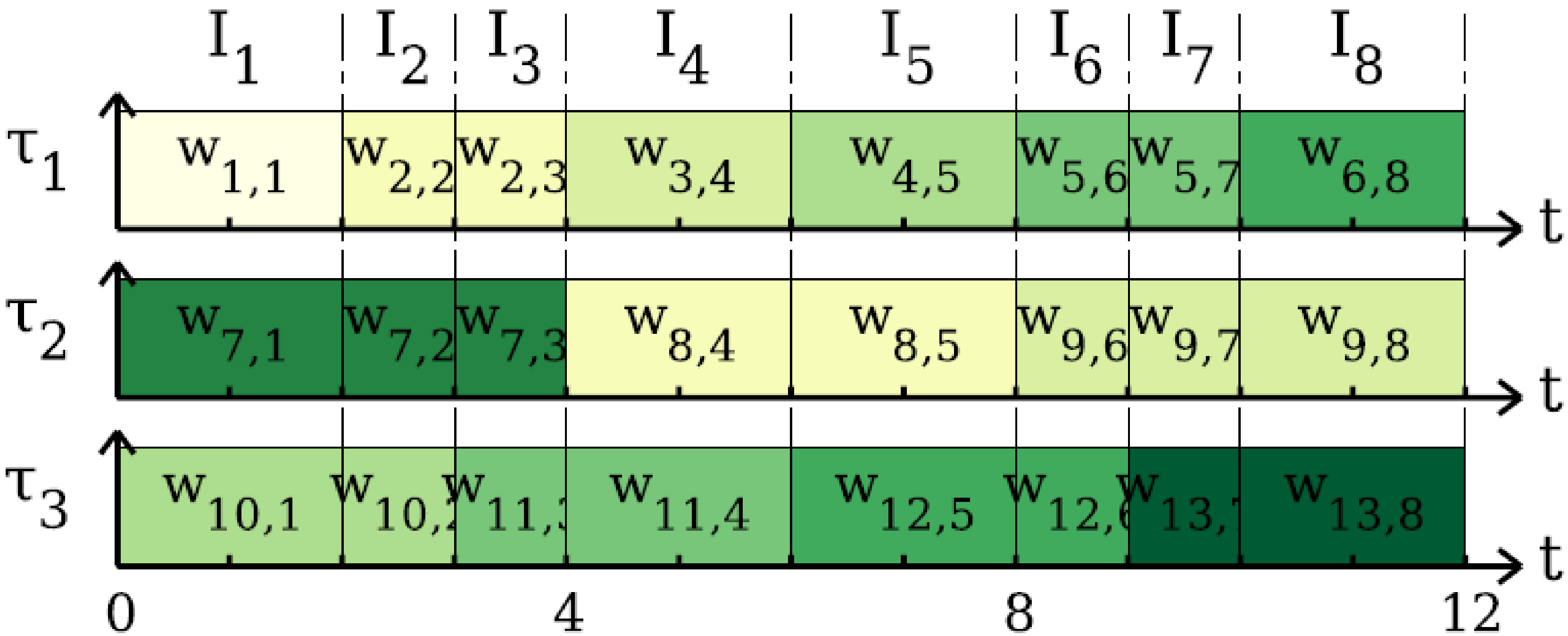
	χ_i	P_i	$C_i(LO)$	$C_i(HI)$
τ_1	LO	2	1.5	1.5
τ_2	HI	4	2	3
τ_3	HI	3	1	2

← J_{LO} (τ_1)

} J_{HI} (τ_2, τ_3)

$U_{HI} = 1.41$

$U_{LO} = 1.58$



LPMC-HI: HI-criticality mode first

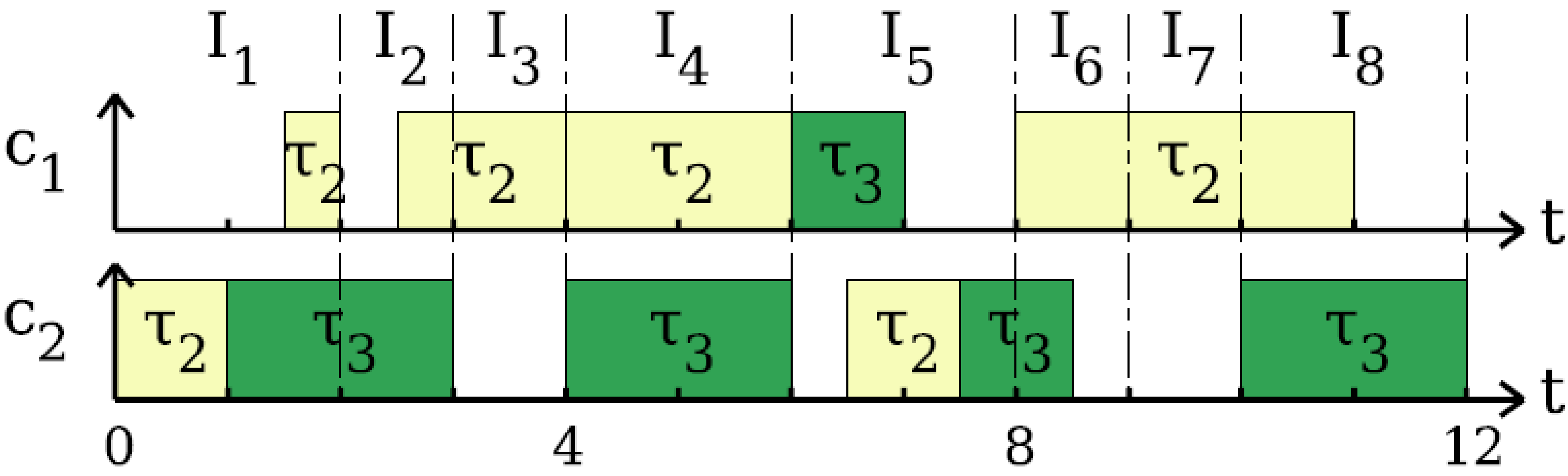
- S_{HI} : temporal schedulability constraint for all jobs
  - Processor maximum utilization: $\sum_{j \in J_k} w_{j,k}^{HI} \leq m, \forall k \in I$
  - No parallel jobs: $0 \leq w_{j,k}^{HI} \leq 1, \forall k \in I, \forall j \in J_\Gamma$
  - Different constraints for the completion
    - HI-criticality jobs $\sum_{k \in E_j} w_{j,k}^{HI} \times |I_k| = C_i(HI), \forall j \in J_{HI}$
    - LO-criticality jobs $\sum_{k \in E_j} w_{j,k}^{HI} \times |I_k| \leq C_i(LO), \forall j \in J_{LO}$
- Objective: prepare the building of S_{LO} in order to maximize the schedulability of J_{LO}
  - Decision variable F_j to account when a LO-criticality job has been completely executed
  - Objective function: maximize $\sum_{j \in J_{LO}} F_j$

LPMC-HI: LO-criticality mode

- Prepare the input for the computation of S_{LO}
  - Execution time of HI-criticality task is reduced to its C_i (LO) values
    - When the HI-criticality task starts does not change
  - The weights of HI-criticality tasks are becoming **constants**: $w_{j,k}^{LO\,'}$
- S_{LO} : temporal schedulability constraints
  - Processor maximum utilization: $\sum_{j_{LO} \in J_k} w_{j,k}^{LO} + \sum_{j_{HI} \in J_k} w_{j,k}^{LO\,'} \leq m, \forall k \in I$
  - Other constraints for the LO-criticality tasks only
    - No parallel jobs
    - Completion of jobs
- No objective function
  - Any feasible solution generates a valid scheduling

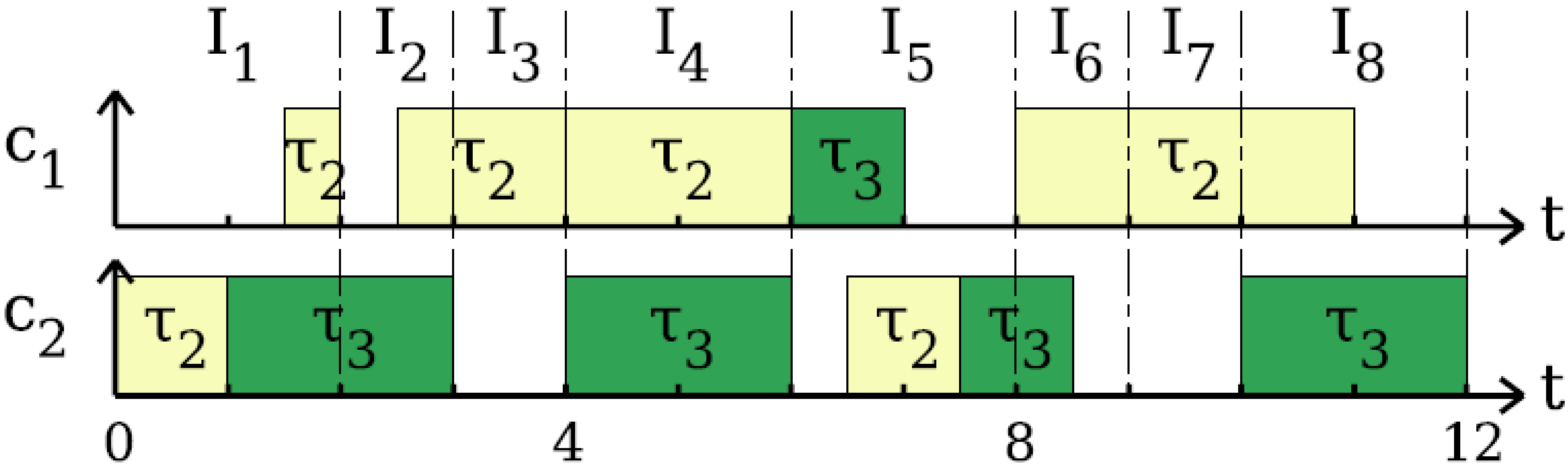
LPMC-HI: computing S_{HI}

- Third and six instances of τ_1 ($P_1 = 2$ and $C_1(LO) = 1.5$)
  - Cannot be completely executed in intervals I_4 and I_8



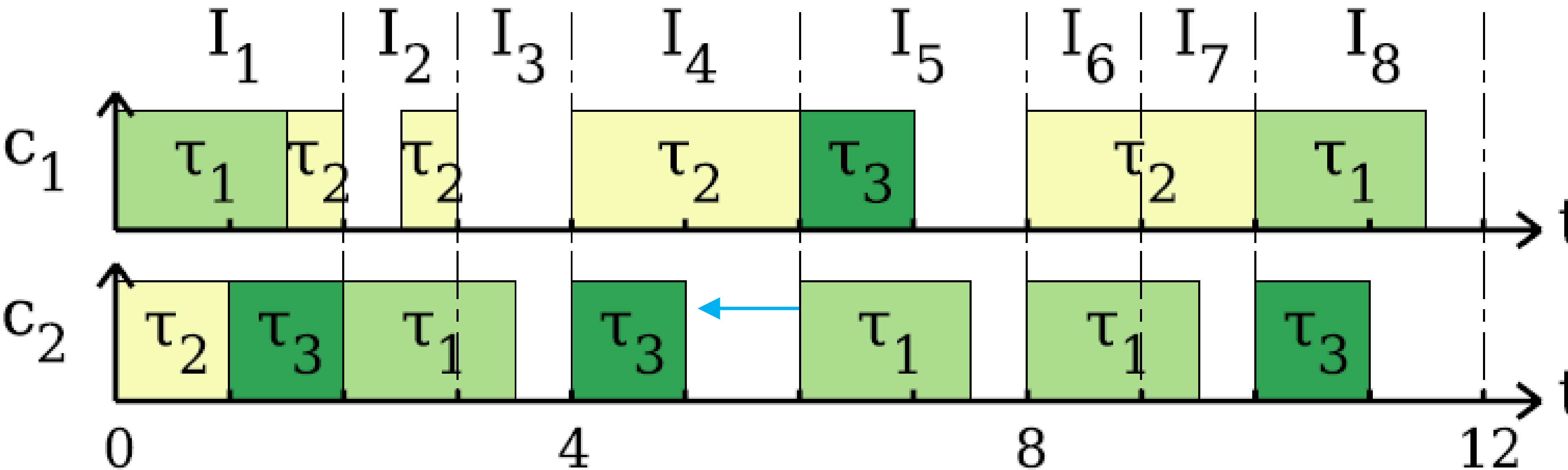
LPMC-HI: problem for computing S_{LO}

- HI-criticality tasks can be concentrated in some particular intervals leading to an unfeasible schedule for S_{LO}
  - Constraints cannot be met in interval I_4



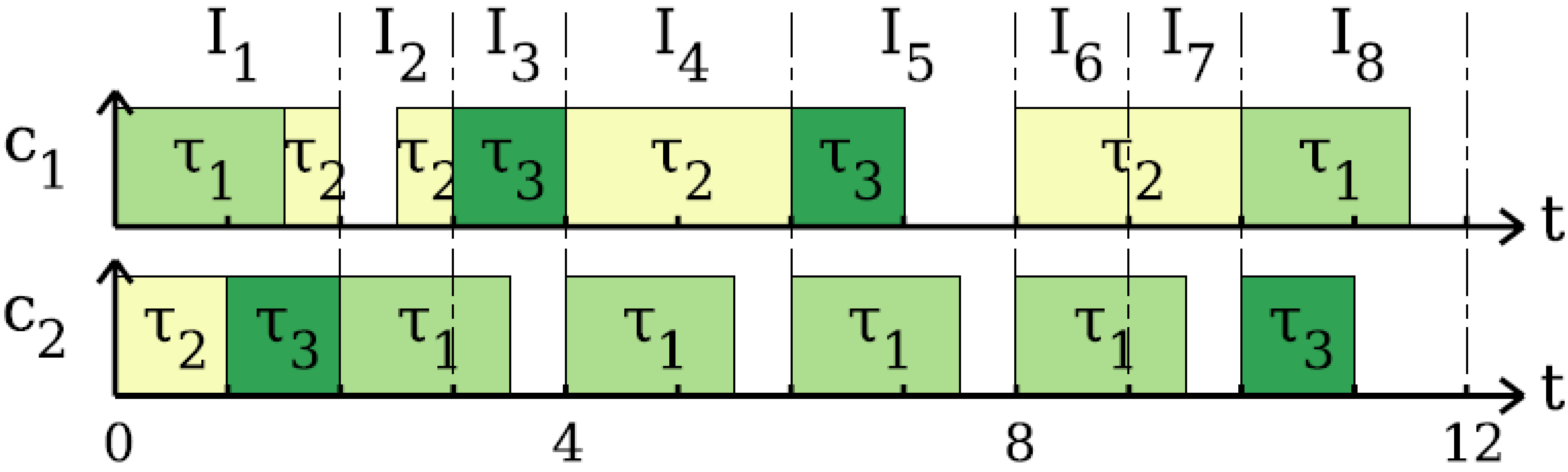
LPMC-HI: problem for computing S_{LO}

- HI-criticality tasks can be concentrated in some particular intervals leading to an unfeasible schedule for S_{LO}
  - Constraints cannot be met in interval I_4



LPMC-HI: problem for computing S_{LO}

- HI-criticality tasks can be concentrated in some particular intervals leading to an unfeasible schedule for S_{LO}
  - Constraints cannot be met in interval I_4



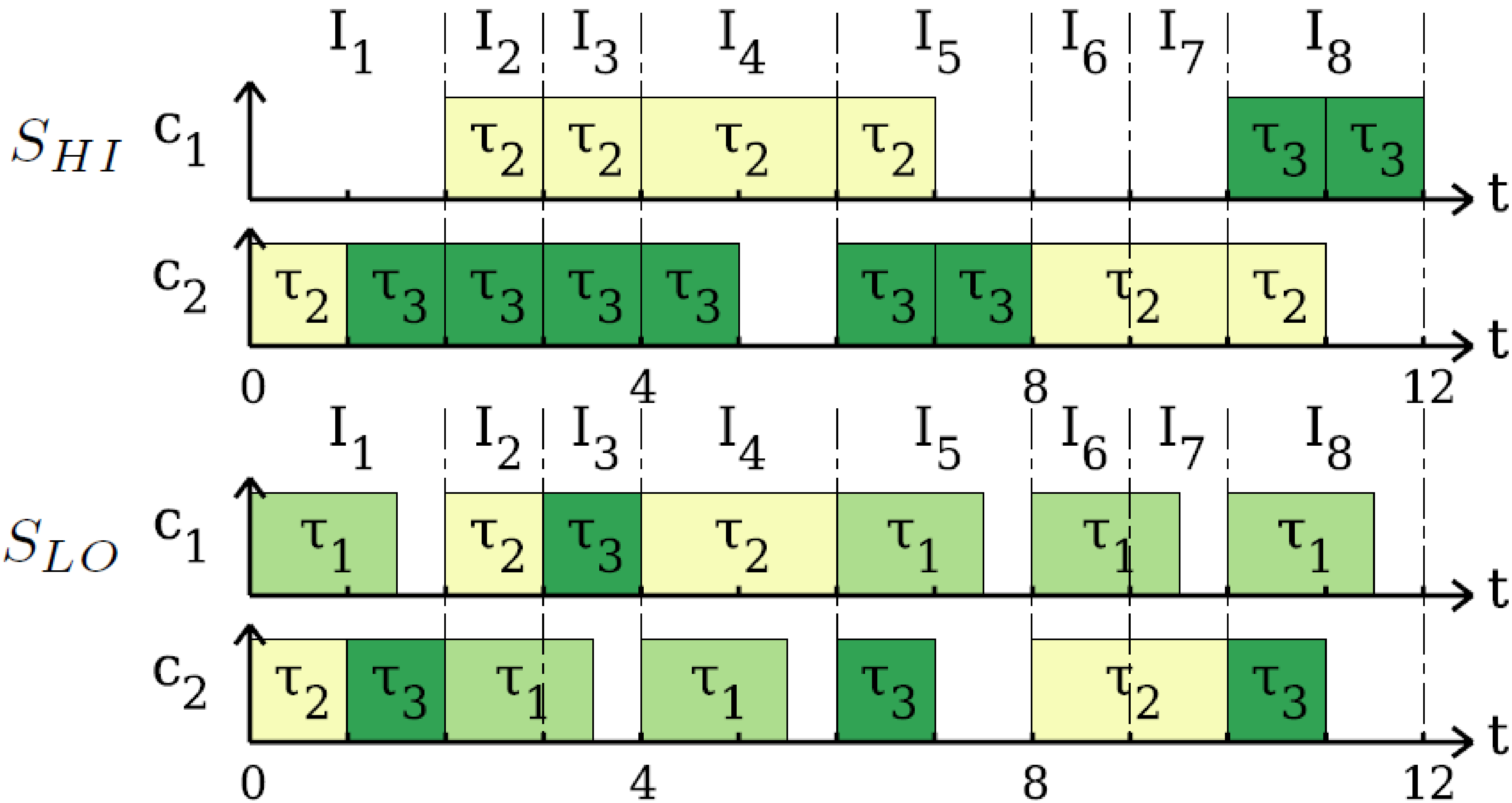
LPMC-Both: both criticality modes

- Goal: improve the schedulability success ratio
  - A single linear program for both S_{HI} and S_{LO}
  - Split each HI-criticality job in two sub-jobs: J_i^{LO} and J_i^{Δ}
    - $w_{j,k}^{\Delta} + w_{j,k}^{LO} = w_{j,k}^{HI}$
  - Temporal schedulability constraints for HI-criticality jobs to compute $w_{j,k}^{HI}$ and for LO-criticality jobs to compute $w_{j,k}^{LO}$

- Precedence constraint to ensure correctness
  - $w_{j,k}^{\Delta}$ must be null till C_i (LO) is not exceeded
  - Prevent sub-jobs from a HI-criticality job to be present in the same interval in S_{LO}
  - In the first interval where C_i (LO) is exceeded, the weight left to J_i^{Δ} is constrained so that a schedule where jobs cannot be executed in parallel can be found: $w_{j,k}^{\Delta} + w_{j,k}^{LO} \leq |I_k|$
  - No constraints in the other intervals

LPMC-Both: scheduling the example

- Both S_{LO} and S_{HI} can be computed



Complexity analysis

- Depends on the number of intervals
- Complexity of LPMC-Both is higher than LPMC-HI
  - Total number of decision variables and constraints increased by:

$$2 \times |I| \times n_{HI}$$

    - n_{HI} : number of HI-criticality tasks
  - Job splitting & precedence constraints

Outline of the talk

- Real-Time scheduling context
  - The Time-Triggered execution model
  - Mixed Criticality systems: two main task models
- Supporting MC within TT using the Vestal task model
  - Building scheduling tables: two solutions based on Linear Programming
- Supporting MC within TT using the elastic task model
  - Maintaining task scheduling consistency: on-line decision algorithm
- Conclusion and future work

Supporting the elastic task model within TT

- 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 its estimated WCET

- In the TT paradigm, visibility dates are predefined
  - Visibility date of data: deadline of the producer
    - A task may only use data whose visibility dates are equals or inferior to its release date
    - To achieve determinism execution behavior
  - The use of stretching factors change the visibility date
    - Inconsistent with the statically defined triggering points
  - On-line decision algorithm to set stretching factor values
    - We assume a dynamic scheduler

Specific notations

- Distinguish between high and low-criticality tasks
  - n_{high} high-criticality task (Γ_{ct}) with a utilization noted U_{high}
  - n_{low} low-criticality tasks (Γ_{nct}) with a utilization noted U_{low}
  - Temporal parameters of a task τ_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

$$1 \leq S_i \leq S_{i,min} \leq S_{i,max}$$

Schedulability analysis

- 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

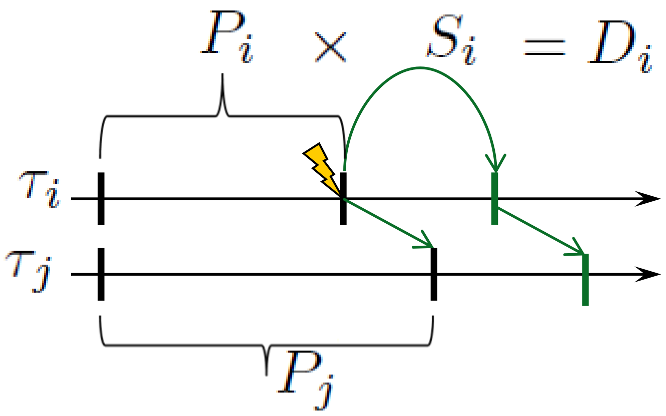
$$u_{i_{min}} \leq \frac{C_i}{S_i \times P_i} \leq u_{i_{max}}$$

- 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

- Two constraints to ensure
  - 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



- 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: Γ_{nct_k}

On-line decision algorithm

- Decision algorithm
  - Assumes that high-criticality tasks have an higher priority
- When it is called?
  - At the beginning of an overloaded situation for the low-criticality tasks
  - When low-criticality tasks have already missed their deadline
    - Within an overloaded situation, where low-criticality tasks were preempted for executing some high-criticality tasks
  - When it is called, we assume that the most important low-criticality task is being executed
- When a stretched low-criticality task finishes, the stretching factor is reset to 1

On-line decision algorithm

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

1: $S_{\Gamma_{nct_k}} \leftarrow ComputeStretching(\tau_i, t, D_i, \Gamma_{nct_k})$;

2: **if** $S_{\Gamma_{nct_k}} > S_{\Gamma_{nct_k,min}}$ **then** Stop Γ_{nct_k} and log the error;
end if

3: $D_i \leftarrow S_{\Gamma_{nct_k}} * P_i$;

4: UpdateReady(τ_i);

for all $\tau_j \in \Gamma_{nct_k} \neq \tau_i$ **do**
  if τ_j is ready **then** RemoveFromReady(τ_j);
  else RemoveFromSleeping(τ_j); **end if**
  $D_j \leftarrow P_j + (D_i - P_i)$;
  if τ_j is finished **then** SetFlag(Stretched); **end if**
  InsertReady(τ_j);
end for

Maintain offsets

5: Call the scheduler;

To ensure that visibility date will be changed

Conclusion

- Adding the support of MC within TT
  - TT: Determinism but low resource utilization in the average case
  - MC: efficient use of processing capabilities in the average case

- Proposal for supporting the Vestal task model within the TT paradigm
  - Two solutions: LPMC-HI and LPMC-Both

- Proposal for supporting a flavor of the elastic task model within the TT paradigm
  - Computation of the minimum required stretching factors
  - Decision algorithm to deal with stretching factors

Future work

- Using the Vestal task model within the TT paradigm
  - Finish the implementation of the proposed solutions
  - Evaluation of their success ratio in scheduling MC job sets

- Using the elastic task model within the TT paradigm
  - 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 TT approach (eXternal-Triggered)

Backup slide

Complexity analysis

- LPMC-HI
  - Total number of decision variables
    - Weights of jobs: $|I| \times n$
    - Decision variable F_j : $|J_{LO}|$
    - Weights of jobs: $|I| \times n_{LO}$

    First LP (weights of jobs and decision variable F_j); Second LP (weights of jobs $|I| \times n_{LO}$)
  - Total number of constraints
    - Number of variables + $|I| + |J_\Gamma|$ (First LP: processor max. capacity, completion) + $|I| + |J_{LO}|$ (2nd LP: processor max. capacity, completion LO)
- LPMC-Both
  - Total number of decision variables increased by $2 \times |I| \times n_{HI}$
    - Job splitting & precedence constraint
  - Total number of constraint
    - For computing S_{LO} : $|I| \times (n+1) + |J_\Gamma|$
    - For computing S_{HI} : $|I| \times (n_{HI}+1) + |J_{HI}|$
    - Precedence constraint: $2 \times |I| \times n_{HI}$

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

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