State of the art of Mixed-Criticality Distributed Systems

Presenting the Jitter-Based Mixed-Criticality Scheduling

Franco Cordeiro

My research

Master in Embedded Systems.

Phd theme: Mixed-Criticality systems for a heterogeneous drone fleet.

Supervised by Laurent Pautet and Samuel Tardieu.

Focus on scheduling of tasks in a system.

Motivation

- Context: IoT and drone fleet
  - Embedded systems;
  - Distributed systems;
  - Real-time restrictions;
  - Cyber-physical systems;
- Objectives:
  - Target distributed heterogeneous system scheduling;
  - Guarantee safety while reducing system oversizing;

Why JMC

K. Lee, M. Kim, H. Kim, H. S. Chwa, J. Lee, J. Lee, and I. Shin. **JMC: Jitter-based mixed-criticality scheduling for distributed real-time systems.** *IEEE Internet of Things Journal, 6(4):6310–6324, 2019.*

- One of the few research works done on distributed heterogeneous system scheduling. (Burns, Alan, and Robert Ian Davis. "Mixed criticality systems-a review:(february 2022)." (2022))
- Scheduling strategy for an IoT mixed-criticality system.
- Promotes a minimum amount of local mode changes instead of a global one.

Safety Critical Systems

A function (or task) is assigned a criticality level according to the severity of a fault on its part.

The criticality level determines the acceptable probability of occurrence of faults (in number per hour).

It determines the development rules to apply according to the criticality level.

Avionic classification

Criticality level	Volume of functions	Consequence	Max # of occurrences
E	5%	None	
D	10%	Minor	10^{-3}/h
C	20%	Major	10^{-5}/h
B	30%	Hazardous	10^{-7}/h
A	35%	Catastrophic	10^{-9}/h

Real-Time Systems

A real-time system consists in one or more sub-system that have to react under specified time requirements to stimuli produced by the environment.

A response after a deadline is invalid even if the response is logically correct.

Missing a deadline is considered as a fault.

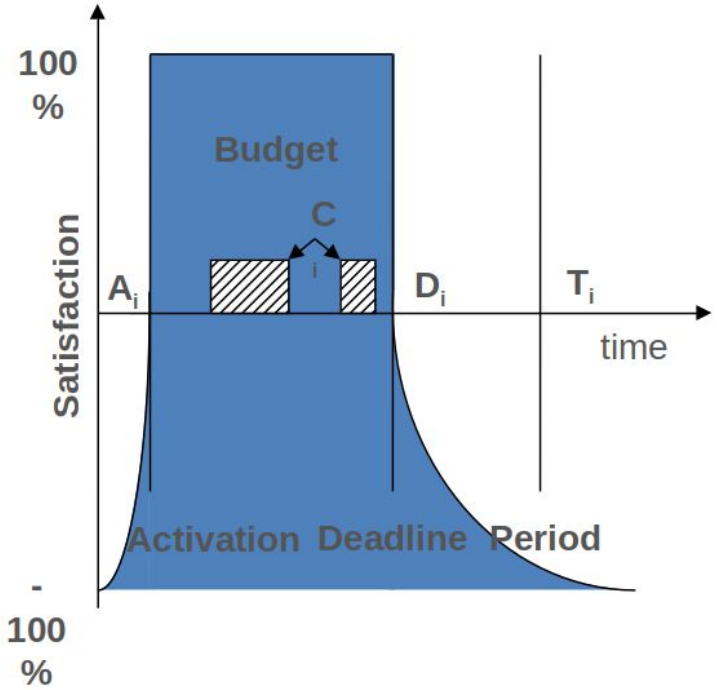
A correct scheduling allocates tasks or functions to processors such that they respect their deadlines.

Finding a correct scheduling under general assumptions on the task set is NP-hard.

Real-Time Systems

For a task t_i with period T_i

- **D_i : Deadline**. For deadline implicit tasks, $D_i = T_i$; usually $D_i <= T_i$; when task t2 depends on task t1, typically $D_1 <= A_2$
- **C_i :** Capacity/**Budget**. Allocated execution time. Usually worst case execution time. But WCET is hard to evaluate;



Mixed Criticality Systems (MCS)

Objective : Execute / mix functions of different criticality, such as survival functions (high critical - HI) and mission functions (low critical - LO).

Traditional critical systems allocate resources in order to guarantee the successful execution of all system functions in the worst case scenario. This results in **oversizing** the architecture (wasted resources).

Approach: Each task has a criticality level and a set of execution parameters per criticality mode.

Low criticality (default) mode: Allocate to HI functions less resources than the worst case scenario (optimistic) would require. Execute LO functions as required.

High criticality mode: LO functions are degraded or stopped as their failure rate is greater and their resources are reallocated to HI functions.

Mode change: occurs when resources are missing for HI functions (a worst case scenario really occurs). This mode change must be seamless so the HI functions still meet their constraints.

Distributed Systems vs monolithic systems

Communication delays are not negligible.

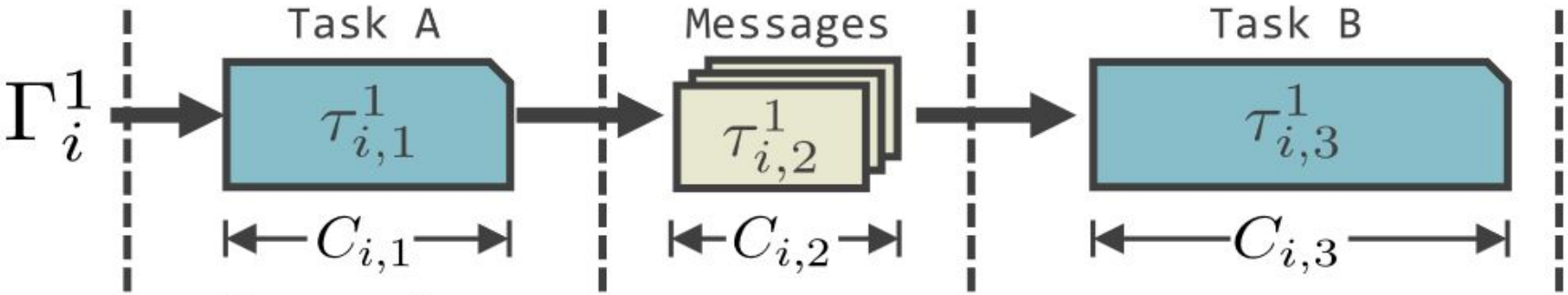
Global (distributed) mode change is prohibitive.

Distributed systems are usually modeled with dependent tasks (flows or DAGs) instead of independent tasks.

Higher unpredictability due to interferences caused by communication channels in addition to local ones.

System overview

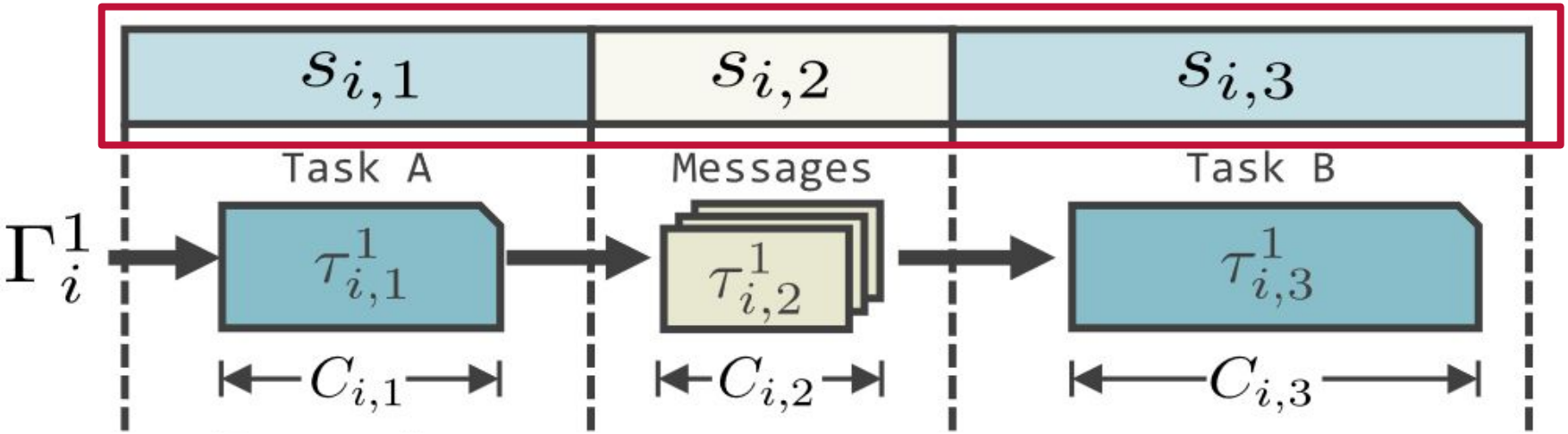
- Multiple computing nodes connected via network links;
- **End-to-end flow:** set of tasks that need to be executed in the same or different computing nodes while transmitting messages in between;
- **Job:** an instance of a flow;



System overview: the flow goes through 3 stages.

System overview

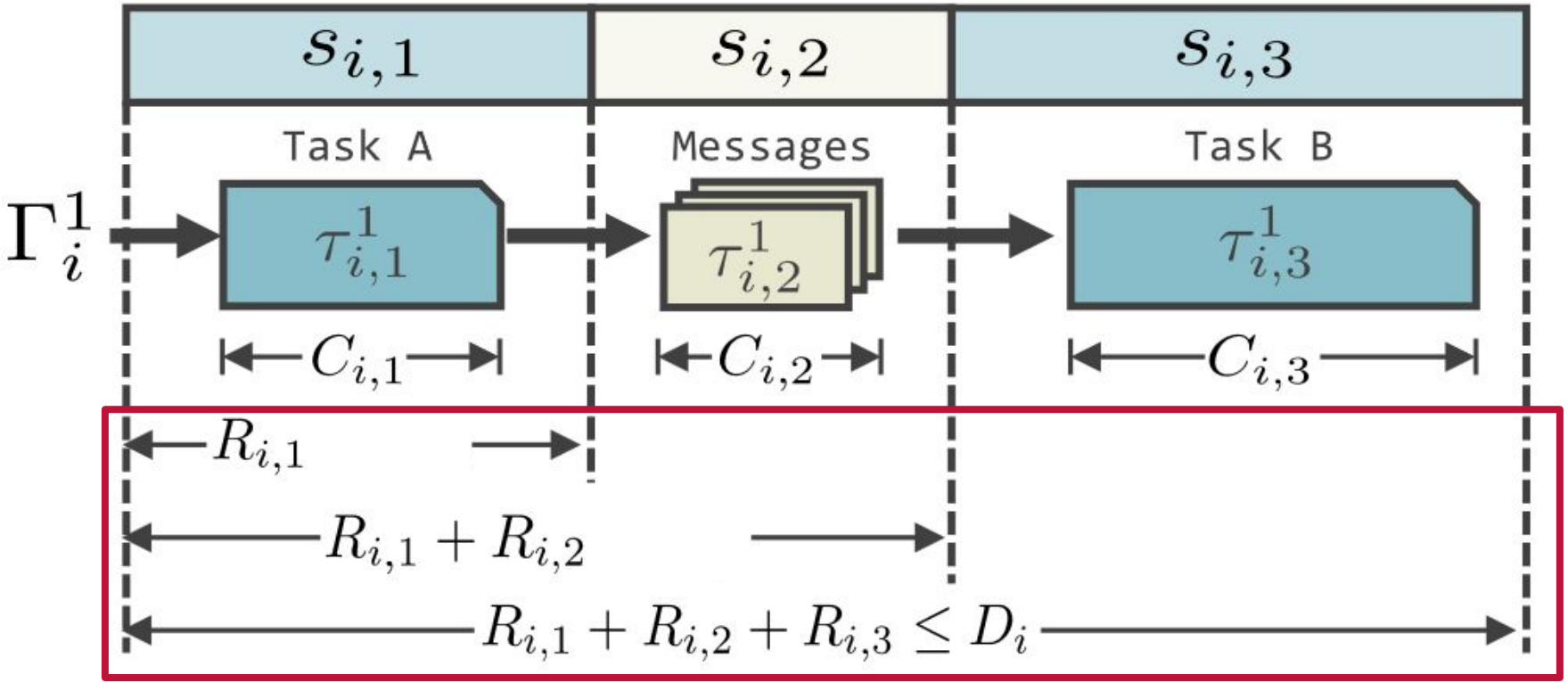
- Multiple computing nodes connected via network links;
- **End-to-end flow:** set of tasks that need to be executed in the same or different computing nodes while transmitting messages in between;
- **Job:** an instance of a flow;
- **Stages:** either a computing node or a network link;



System overview: the flow goes through 3 stages.

System overview

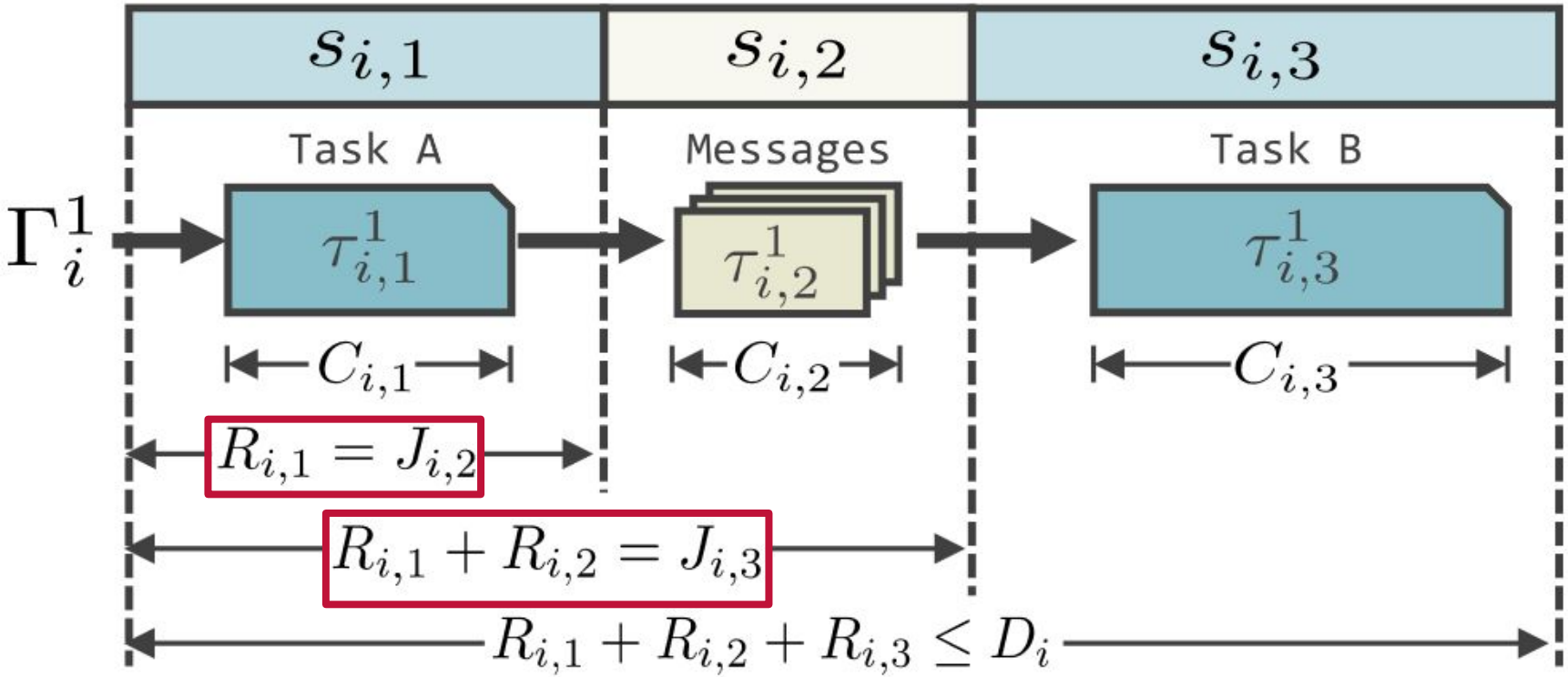
- Multiple computing nodes connected via network links;
- **End-to-end flow:** set of tasks that need to be executed in the same or different computing nodes while transmitting messages in between;
- **Job:** an instance of a flow;
- **Stages:** either a computing node or a network link;
- **Response time:** time it takes for a task or transmission to finish its execution;
- **Worst case response time(WCRT):** response time of a task or transmission in the worst possible scenario. The flow WCRT must be inferior to its deadline;



System overview: the flow goes through 3 stages.

System overview

- Multiple computing nodes connected via network links;
- **End-to-end flow:** set of tasks that need to be executed in the same or different computing nodes while transmitting messages in between;
- **Job:** an instance of a flow;
- **Stages:** either a computing node or a network link;
- **Response time:** time it takes for a task or transmission to finish its execution;
- **Worst case response time(WCRT):** response time of a task or transmission in the worst possible scenario. The flow WCRT must be inferior to its deadline;
- **Release jitter:** the duration between the release of a flow and its arrival at a specific stage;



System overview: the flow goes through 3 stages.

Assumptions

- Tasks have the same WCET for all modes;
- LO tasks are dropped when the mode changes to HI;
- Stages are either:
  - Monocore computing nodes with fixed priority scheduling;
  - Direct point-to-point network links;
- Each stage determines whether a mode change is required or not;

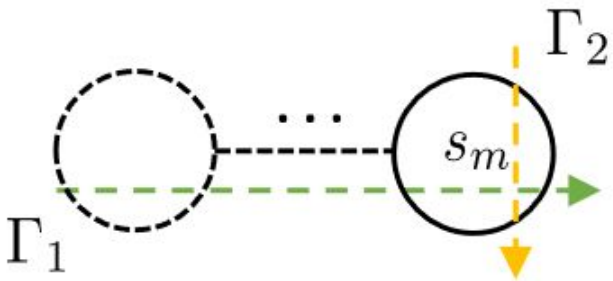
Goals and pessimistic approach

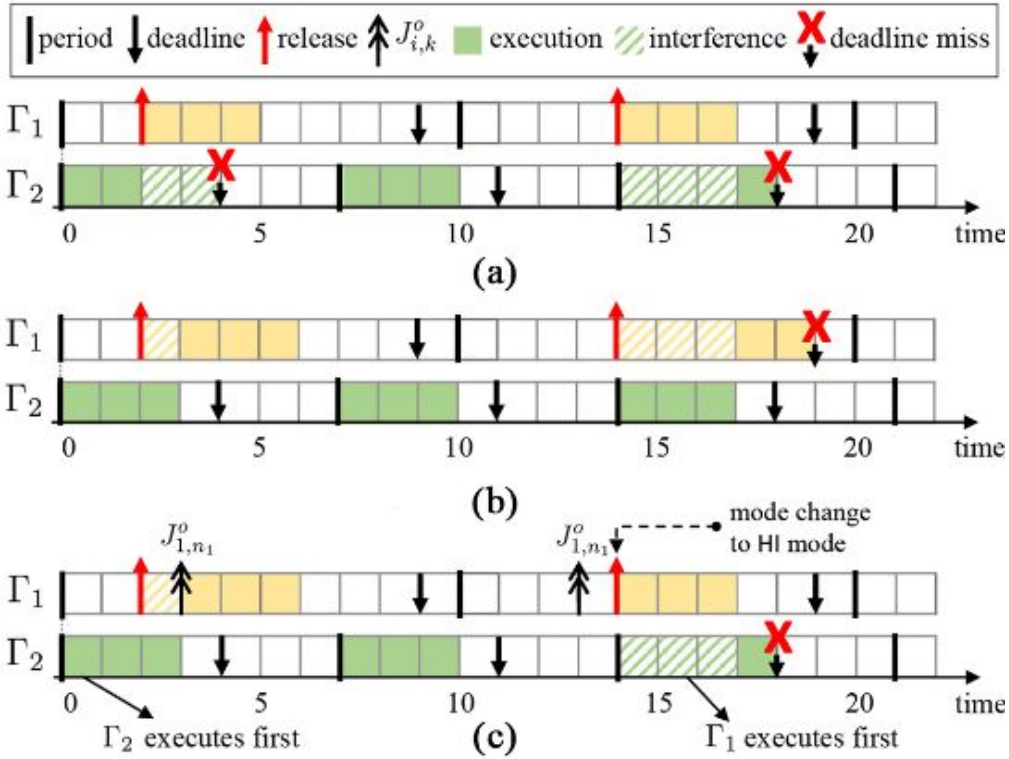
- Goals:
  - **G1:** Guarantee that all jobs of HI flows meet end-to-end deadlines.
  - **G2:** Maximize the number of jobs of LO flows that meet end-to-end deadlines.
- Holistic analysis of WCRT: sum of the WCRT in every stage of a flow.
- Pessimism on holistic analysis:
  - **Per-Flow Minimum Interarrival Time:** first job arrives with maximum jitter, while subsequent jobs arrive with minimum jitter.
  - **Per-Stage Critical Instant:** Every higher priority jobs arrive simultaneously with maximum jitter.
  - **End-to-End Delays:** addition of extremely rare conditions on every stage.

Stage mode change

- Each task or message has an optimistic WCRT and a pessimistic one;
- If the optimistic WCRT is violated in a stage, the next stage in a flow changes its mode to HI;
- Each stage can detect the necessity of a mode change by looking at the release jitter of a job;
- **LO to HI:** A jitter threshold is defined to detect the necessity of a mode change to HI;
- **HI to LO:** Once every HI flow that arrived with a jitter greater than the threshold has finished its execution, the mode is changed to LO again;

Flows	T_i	C_{i,n_i}	D_i	L_i
Γ_1	10	3	9	HI
Γ_2	7	3	4	LO





Scheduling of the example: (a) CA-DM. (b) EDF. (c) JMC.

Mode scheduling

WCRT is calculable if the flows have a fixed priority (Heuristic analysis).

When a mode change occurs and LO tasks are dropped, the WCRT decreases.

The priority assignment of both modes must guarantee that the order of priority between tasks of same criticality remains the same.

The jitter threshold must be big enough to minimize the number of mode changes, but small enough to guarantee the execution of HI flows.

Lazy Policy

Procrastinates mode change as much as possible.

Maximizes the number of schedulable LO flows when the WCRT analysis of each stage is pessimistic.

Risks having many mode changes at the end.

Uses the largest possible value:

$$J_{i,k}^{o} = D_i - R_{i,k}^{*}(\mathrm{LO}) - \sum_{k+1 \le m \le n_i} R_{i,m}^{*}(\mathrm{HI})$$

Proactive policy

Tries to minimize the number of LO jobs affected.

Maximizes the number of schedulable LO flows when the analysis of the worst case response time is accurate.

Therefore, if the upper bound of LO flows affected is equal to the actual number, the proactive policy achieves G2.

Proactive policy

Calculates the number of LO flows affected by mode change in a stage:

$$\delta_{i,k} = \sum_{\Gamma_j \in \Gamma(\mathrm{LO}) \text{ and } s_{i,k} \in S_j} \left\lceil \frac{R^*_{i,k}(\mathrm{HI})}{T_j} \right\rceil$$

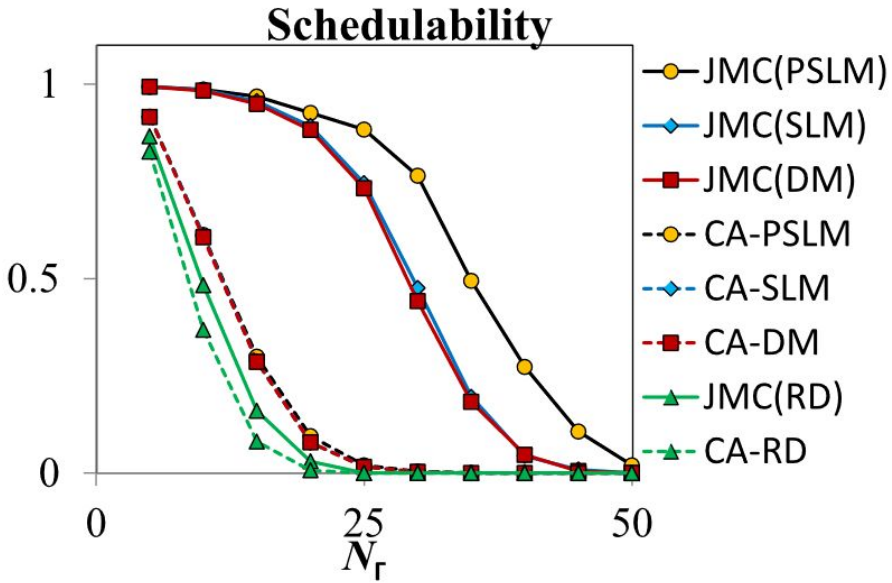
The threshold is then calculated by:

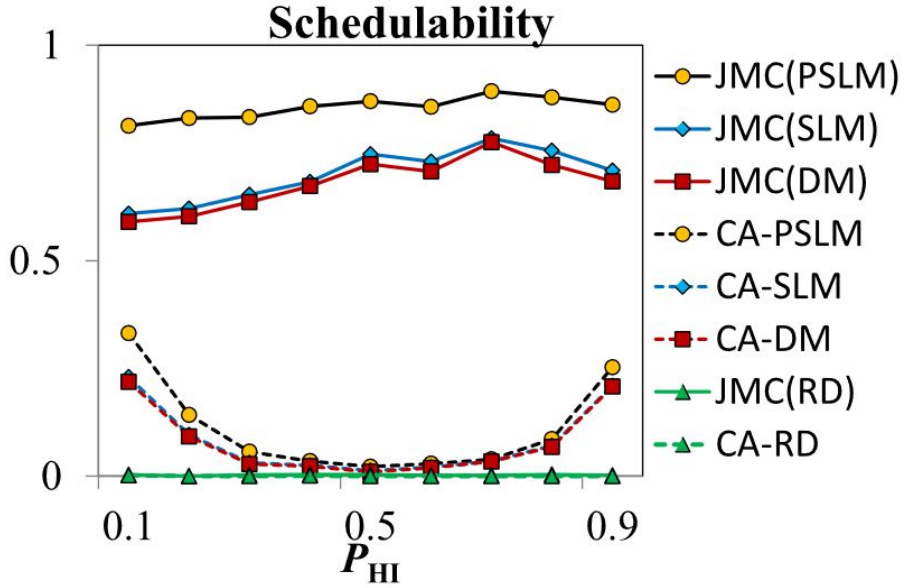
$$J^o_{i,k} = D_i - \sum_{\delta_{i,r} < \delta_{i,k}} R^*_{i,r}(\mathrm{HI}) - \sum_{\delta_{i,t} \geq \delta_{i,k}} R^*_{i,t}(\mathrm{LO})$$

Performance comparison

- 16 computer node connected in a 4x4 grid
- Tested JMC with the following fixed priority algorithms:
  - RD: random;
  - DM: deadline monotonic;
  - SLM: static laxity monotonic;
  - PSLM: per stage static laxity monotonic ([(Di - Ci)/ni]).
- Compared with Criticality Aware policies
  - RD
  - DM
  - SLM
  - PSLM

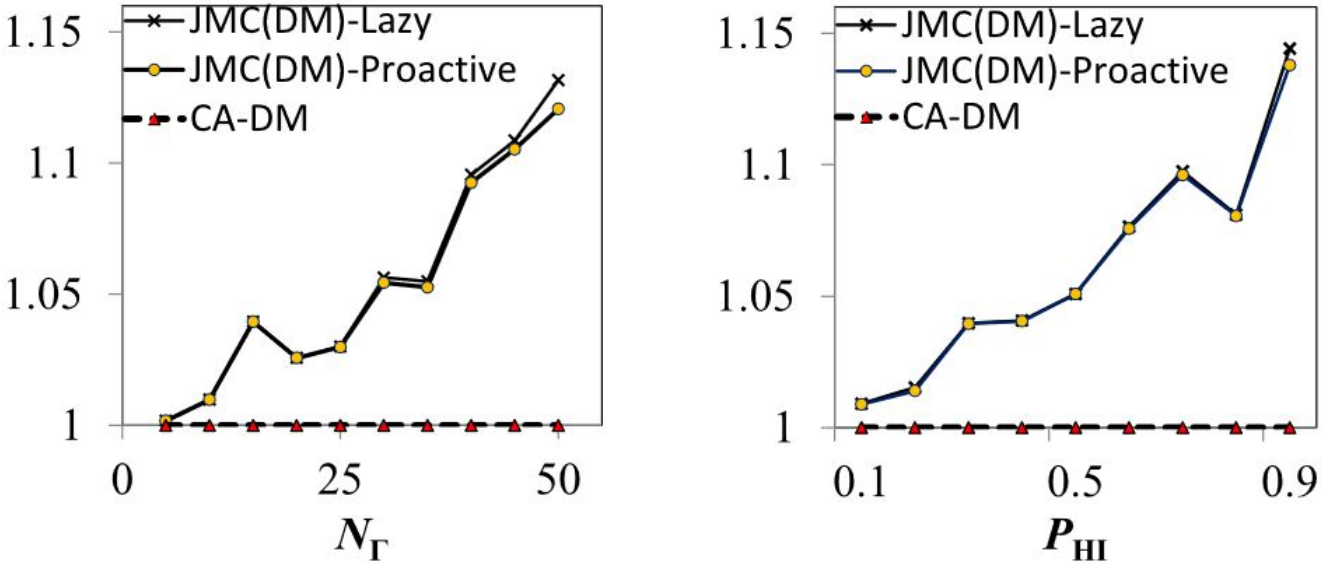
Better schedulability





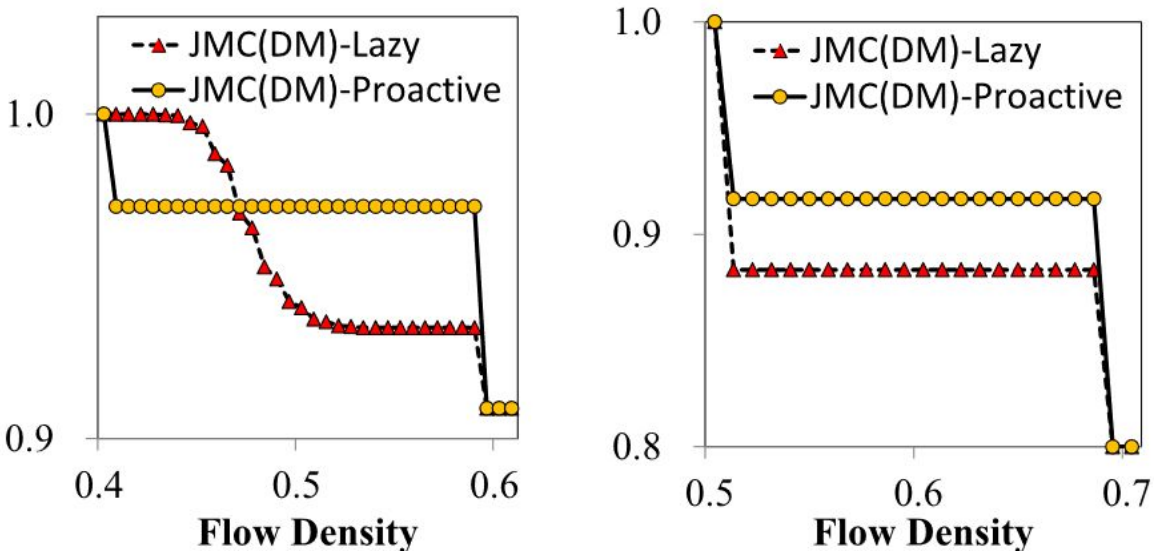
Schedulability in fraction of schedulable flow sets. Nr is the number of flows in a set. PHI is the probability of a flow to have its criticality set to HI.

Less LO flows dropped



Schedulability of LO flows normalized to that of CA-DM. Nr is the number of flows in a set. PHI is the probability of a flow to have its criticality set to HI.

Lazy is better than Proactive on pessimistic analysis



Normalized number of schedulable LO jobs with nonharmonic and harmonic period sets

Conclusion

JMC produces better results than the existing criticality-aware fixed priority assignment scheduling schemes.

Limited system model:

- ➢ Identical WCET for all criticality modes;
- ➢ Restricted to monocore and two criticality levels;
- ➢ Restricted to linear flows.