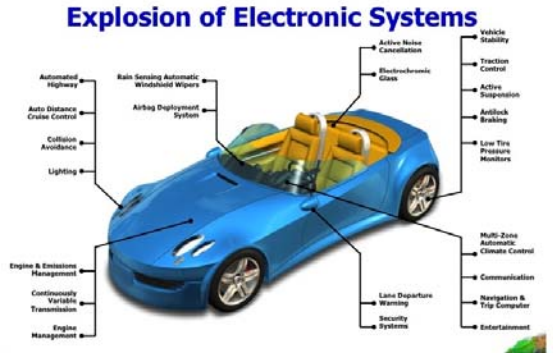
7.d Mixed-criticality systems

Where we see how the want of more-for-less has entered the high-integrity domain, causing tasks with different levels of criticality to be integrated in a mixed-criticality (real-time) system

Background /1

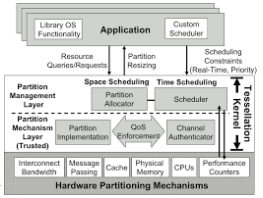
- Critical systems are those that perform essential services
  - When required, they have to run with high assurance: should they not, serious consequences would follow
  - Previously they were dedicated (for HW) and specialized (for SW)
    - Few, sparse, and nearly invisible to the public eye
  - *Isolation* is conservative, it may waste resources to warrant integrity
- Digital transformation wants far greater unitary functional value in those systems
  - *Integration* is pragmatic, it wants more value for less resource usage
  - Not all functions equally essential: some only serve competitive edge (e.g., comfort over safety)
- Tension builds between ***integration*** and ***isolation***

An example of digital transformation

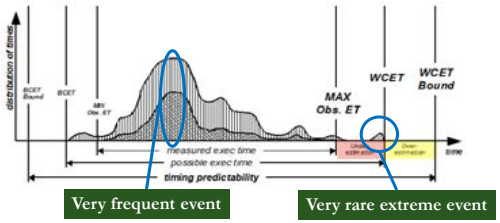


Premise /2

- Isolation makes *static allocations*, with *conservative margins* to mitigate the uncertainty of extreme events
  - Conservative margins are wasteful if the worst-case profile has an extreme tail
    - Very far to the right of the average case
- The baseline approach is known as **Time and Space Partitioning**
  - It warrants isolation via a *resource scheduling* hypervisor



The consequence of conservatism



- Budgeting for the rare extreme here would cost 240% more than provisioning for the average case
- You may not want to budget for the WC statically, but you must be able to sustain it when it happens: *something's gotta give* in that case …

Premise /3

- Well-behaved integration may reduce waste
  - Tasks with different levels of criticality might be allowed to co-exist under strict safeguarding guarantees
  - Main goal (in this line of research) is maximum use of CPU
- Tasks with higher integrity requirements (*high criticality*) must be guaranteed up to their worst case, but their *default* allocation covers only the *high watermark*
  - This is the central tenet of **Mixed-Criticality Systems**
  - When a Hi-crit job executes above average, a **mode change** occurs, which "adjusts things"
    - Thereafter, all Hi-crit tasks retain their guarantees
    - Lo-crit tasks lose them (they are held up), until normality is restored

Vestal's initial vision of MCS (2007)

- The system is single-core
- Tasks are divided in multiple *criticality-based* groups
  - A *mode* attribute $L_i \in \{Lo, \dots, Hi\}$ is attached to each task τ_i, determining its budget allocation
  - Hi-crit tasks are given a *high conservative margin* over their measured WCET
  - Lo-crit tasks have *no* margin
  - Any task can use the unclaimed margin, but only Hi-crit tasks can claim it
- The RTA for this case becomes

$$R_i = C_i(\boldsymbol{L_i}) + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\boldsymbol{L_j})$$

  - Each task is assumed to contribute its per-criticality (L) allocation
- Priority and criticality do *not* coincide: we need a *priority assignment scheme* that serves the MCS intent

Vestal's experimental evidence

			Workload 1		
task	T_i	L_i	measured	allocated	margin
P1 40hz	25	B	1.06	1.4	32.1%
P1 20hz	50	B	3.09	3.9	26.2%
P2 20hz	50	B	2.7	2.8	3.7%
P3 20hz	50	B	1.09	1.4	28.4%
P4 40hz	25	A	0.94	1.1	17%
P4 20hz	50	A	1.57	1.8	14.6%
P4 10hz	100	A	1.68	2.0	19%
P4 5hz	200	A	4.5	5.3	17.8%
P5 20hz	50	B	2.94	3.7	25.9%
P5 10hz	100	B	1.41	1.8	27.7%
P5 5hz	200	B	6.75	8.5	25.9%
P6 20hz	50	D	5.4	5.4	0%
P6 5hz	200	D	2.4	2.4	0%
P7 20hz	50	D	0.94	1.3	38.3%
P7 5hz	200	D	1.06	1.5	41.5%
P8 40hz	25	D	2.28	2.3	0.9%
P8 10hz	100	D	4.75	4.8	1.1%
P8 5hz	200	D	12.87	13	1%
P9 10hz	100	D	0.47	0.6	27.7%
PA 20hz	50	C	1.24	1.9	53.2%
PB 20hz	50	D	1.62	2.4	48.1%
utilization			80.4%	93%	21.4%

Average

Table 1: Example Multi-Criticalty Workload

$$\boldsymbol{margin} = \frac{allocated - measured}{measured}$$

	Workload 1	
method	Δ^*	increase
deadline monotonic priority traditional analysis	1.08	–
deadline monotonic multi-criticality analysis	1.20	11%
multi-criticality Audsley's multi-criticality analysis	1.20	11%
transformed & deadline monotonic multi-criticality analysis	1.20	11%
transformed & multi-criticality Audsley's multi-criticality analysis	1.20	11%

Table 2: Comparative Evaluation Results

Δ^* is the largest simultaneous increase in the budget allocation of *all* tasks (over the *measured* bound) that would preserve *overall* feasibility

The MCS solution assures a 20% margin *without* wastage

Immediate ramifications

- EDF does *not* dominate FPS for systems with criticality levels
  - Feasible systems can be constructed that EDF is unable to schedule
- The MCS model of (constrained-deadline) sporadic task may be formalized as $(T^{\rightarrow}, D, C^{\rightarrow}, L)$ such that L is a set of criticality levels $\{\dots, L_i, L_j, \dots\}$ where

$$L_j > L_i \Longrightarrow C(L_j) \geq C(L_i), T(L_j) \leq T(L_i)$$

  - *The higher the task's criticality, the larger the guarantee above its default allocation*
  - Most commonly, $L = \{Lo, Hi\}$ and $T^{\rightarrow} = T$
- **The solution rests on an effective (fixed) priority ordering**
  - First, order all Hi-crit and Lo-crit tasks with deadline-monotonic ordering
  - At each step, first test the lowest-priority Lo-crit task (Audsley's style)
    - If feasible, it takes that priority
    - Else, try the next task; if none is feasible, failure
  - This logic assures best guarantees for Hi-crit tasks

Adaptive Mixed Criticality (2012)

- Approach proposed by Baruah, Burns and Davis
  - System model still single-core, FPS assumed
- To achieve higher average utilization, WCET allocation is not static
  - When a Hi-crit job runs past its Lo-crit budget, a **mode change** triggers
  - To safeguard Hi-crit tasks, all Lo-crit tasks are (temporarily) suspended
- Three distinct feasibility conditions

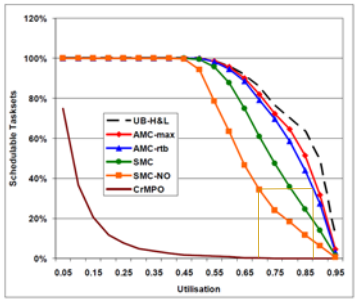
Lo-crit mode: $R_i(Lo) = C_i(Lo) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(Lo)}{T_j} \right\rceil C_j(Lo)$

Hi-crit mode: $R_i(Hi) = C_i(Hi) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(Hi)}{T_j} \right\rceil C_j(Hi)$

Lo-2-Hi: mode: $R_i^* = C_i(Hi) + \sum_{j \in hpH(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j(Hi) + \sum_{k \in hpL(i)} \left\lceil \frac{R_i(Lo)}{T_k} \right\rceil C_k(Lo)$

  - The Lo-crit tasks are (pessimistically) assumed to contribute their maximum interference before being suspended

Asserted benefits



Approaches
UB-H&L: theoretical upper bound
AMC-max: adaptive mixed-criticality (minor tweak over base AMC)
AMC-rtb: adaptive mixed-criticality (base method)
SMC: as Vestal, but with mode-change monitoring
SMC-NO: Vestal's original approach
CrMPO: priorities assigned in order of criticality

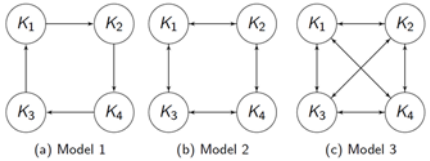
20 tasks per task set, an average of 50% tasks assumed Hi-crit, and $C(Hi) = 2 \times C(Lo)$

What with multicores?

- Higher functional value if Lo-crit tasks could migrate instead of being suspended
  - This requires partitioned scheduling and *per core* criticality mode
  - Hi-crit tasks statically assigned to a core
  - The Lo-crit tasks feasible in Hi-crit mode are statically assigned
  - The Lo-crit tasks that would be abandoned on one core and could fit feasibly on another core, are allowed to migrate to it
  - Residual Lo-crit tasks marked "*expendable*"
- Only a small fraction of cores is assumed to enter Hi-crit mode simultaneously
  - The system should be kept feasible up to that limit
- Solution in three mutually-dependent parts
  - Partition tasks, determine allowable migrations, assign priorities

Xu & Burns (2019) /1

- 3 models of migration for a quad-core processor



  - **Model 1**: each core has one migration route
  - **Model 2**: each core has two migration routes
  - **Model 3**: each core allows migration to all other cores

Xu & Burns (2019) /2

- Order tasks by decreasing criticality
- Use one of (First-Fit, Best-Fit, Worst-Fit) bin-packing for task-to-core assignment
- Use Audsley's algorithm to assign per-core priorities
  - If a Hi-crit task is not feasible on one core, try it on another core
  - If a Hi-crit task cannot be feasibly assigned, then **failure**
  - If a Lo-crit task is not feasible, pick the highest-priority Lo-crit task on that core and try a *migration route* for it
  - If that fails, try the next Lo-crit task down: if any Lo-crit task remains unassigned, mark it expendable
- The system needs to be studied *before* and *after* mode change
  - Dependent on how many cores can enter Hi-crit mode simultaneously
  - We look at the *1-mode-change* case only: the others can be built analogously

Xu & Burns (2019) /3

- **Before** mode change (*steady mode*), core K_s hosts some Hi-crit tasks, some Lo-crit tasks, and some Lo-crit "*can migrate*" tasks

$$R_i(Lo) = C_i(Lo) + \sum_{j \in hp(i)} \left\lceil \frac{R_i(Lo)}{T_j} \right\rceil C_j(Lo)$$

- **After** mode change ($L_i > Lo$) in core K_s, with migration route to core K_t
  - Core K_s sheds its "*can migrate*" Lo-crit tasks ($\boldsymbol{M_{K_s}}$), which contribute their maximum interference before going

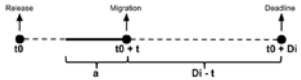
$$R_i(L_i) = C_i(L_i) + \sum_{j \in hp(i), K_s} \left\lceil \frac{R_i(L_i)}{T_j} \right\rceil C_j(L_j) + \sum_{\omega \in hp(i), \boldsymbol{M_{K_s}}} \left\lceil \frac{\boldsymbol{R_i(Lo)}}{T_\omega} \right\rceil C_\omega(Lo)$$

Xu & Burns (2019) /4

- **After** mode change, in core K_t with migration from core K_s
  - Core K_t will have to schedule the incoming Lo-crit tasks

$$R_i(Lo) = C_i(Lo) + \sum_{j \in hp(i), K_t} \left\lceil \frac{R_i(Lo) + \boldsymbol{J_j}}{T_j} \right\rceil C_j(Lo)$$

  - Any "can migrate" task τ_j will carry residual work $(C_j - a)$ with relative deadline $(D_j - t)$ to core K_t

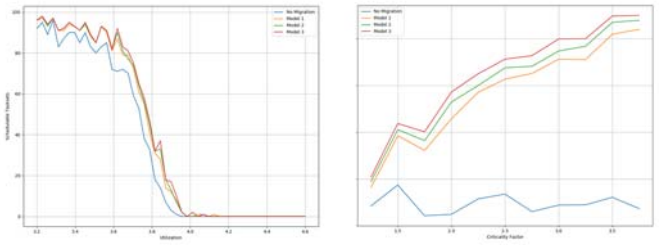


  - Any such task τ_j will suffer release jitter $J_j \leq R_j(Lo) - C_j(Lo)$

Performance evaluation /1

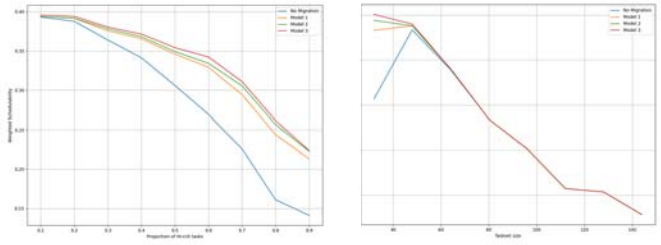
- Analysis scenarios (no real execution)
  - Percentage of schedulable task sets over increasing utilization
    - No migration (AMC) vs. model 1, model 2, model 3
  - Weighted schedulability as a function of two factors
    - How much one approach dominates the other(s)
- Over varying parameters
  - Log-uniform period distribution
  - Size of task set
  - Plot 1: ratio of Hi-crit tasks in task set
  - Plot 2 (criticality factor): Hi:Lo inflation rate in demand

Performance evaluation /2



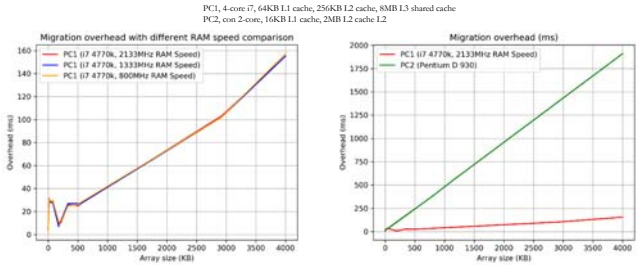
Results reproduced by:
https://github.com/cornacchia/py-xu-burns-2019-rta

Performance evaluation /3



- Migration affords more schedulable utilization
  - At least 10% more feasible task sets for $4 \geq U > 3.8$, and task sets cardinality no larger than 50 tasks
  - The gain increases as the proportion of Hi-crit tasks or the criticality margin grow
  - Model 1, by far the simplest, suffices

Migration costs are not negligible



Minimal Linux, 10k R/W random access ops on variable-size array (0.4 kB – 4 MB in 0.4 kB increments), job migration every even iteration

Summary

- Digital transformation wants real-time systems to embed an ever increasing number of value-added software functions
  - Some such functions are of *high integrity* and must be given high assurance
  - Other functions are less critical, but we want to deploy them in the same processor as the other functions to have more functional value per unit of computation
- This need has originated *mixed-criticality systems*
  - We have examined approaches that give sufficient assurance of *time isolation* while achieving high schedulable utilization

Selected readings

- S. Vestal (**2007**)
  Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance
  DOI: 10.1109/RTSS.2007.47
- H. Xu, A. Burns (**2019**)
  A semi-partitioned model for mixed criticality systems
  10.1016/j.jss.2019.01.015