Revisiting the Computational Complexity of Mixed-Critical Scheduling

Rany Kahil, Peter Poplavko, Dario Socci, Saddek Bensalem

Verimag, University of Grenoble Alps

December 05, 2017

Outline

Problem Formulation

- A **job** J_j is characterized by a 5-tuple $J_j = (j, A_j, D_j, \chi_j, C_j)$
  - $j \in \mathbb{N}_+$ is a unique index
  - $A_j \in \mathbb{N}$ is the arrival time, $A_j \geq 0$
  - $D_j \in \mathbb{N}$ is the deadline, $D_j > A_j$
  - $\chi_j \in \{\mathrm{LO}, \mathrm{HI}\}$ is the job's criticality level
  - $C_j \in \mathbb{N}_+^2$ is a vector $(C_j(\mathrm{LO}), C_j(\mathrm{HI}))$ where $C_j(\chi)$ is the WCET at criticality level χ.
- An **instance** of the scheduling problem is a set of K jobs $\mathbf{J}$
- A **scenario** of an instance $\mathbf{J}$ is a vector of execution times of all jobs $c = (c_1, c_2, \ldots, c_K)$

Problem Formulation

- The **criticality of a scenario** c is
  - LO if $c_j \leq C_j(\mathrm{LO}), \;\; \forall j \in [1, K]$
  - HI otherwise
- A **mode switch** occurs at the first time instant where a HI job J_j exceeds $C_j(\mathrm{LO})$
- A scenario c is **basic** if
  - $\forall j = 1, \ldots, K \quad c_j = C_j(\mathrm{LO}) \vee c_j = C_j(\mathrm{HI})$
  - *LO* **scenario** $\forall j = 1, \ldots, K \quad c_j = C_j(\mathrm{LO})$
- A **schedule** $\mathcal{S}$ of a given scenario c is a mapping: $\mathcal{S} : T \mapsto \widehat{\mathbf{J}}_m$ where T is the physical time and $\widehat{\mathbf{J}}_m$ is the family of subsets of $\mathbf{J}$ that contains all subsets $\mathbf{J}'$ of $\mathbf{J}$ such that $|\mathbf{J}'| \leq m$ where m is the number of processors.

Problem Formulation

- A **scheduling policy** for an instance **J** specifies deterministically which job to execute at each time instant
- A scheduling policy is **correct** for the given problem instance if the following conditions are respected
  - If all jobs run at most for their LO WCET then all jobs must terminate before their deadline
  - If at least one job runs for more than its LO WCET then all critical jobs must terminate before their deadline
- A policy is said to be **work-conserving** if it never idles the processor if there is pending workload
- An instance **J** is **MC-schedulable** if there exists a correct scheduling policy for it

Scheduling Policies (Dual-Criticality Systems)

- **Fixed Priority (FP)** is a work-conserving policy that is characterized by a priority table defining a total order relation between the jobs
- **Fixed Priority per Mode (FPM)** is a mode-switched policy that came as an extension of fixed-priority for mixed criticality systems by using two prioirty tables one for each mode
- **Static Time-Triggered Table per Mode (STTM)** where there are only two time-triggered tables - one per mode

The Complexity of MC-Scheduling

- MC-schedulability is NP-hard in the strong sense, even when all release times are identical and there are only two criticality levels [Baruah et al. (2012)]
- The problem of deciding MC-schedulability for L criticality levels is in NP when L is a constant [Baruah et al. (2012)](revisited)
  - Optimal schedule can be checked for correctness in polynomial time
  - **Revisited lemma.** If an instance is MC-schedulable, then there exists an optimal online scheduling policy that preempts each job j only at time points t such that at time t either some other job is released, or j has executed for exactly $C_j(i)$ units of time for some $1 \leq i \leq L$

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_1 for execution

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_1 for execution
- According to the revisited lemma job J_1 need not be preempted before time $t = 5$

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_1 for execution
- According to the revisited lemma job J_1 need not be preempted before time $t = 5$
- In the interval $[5, 11)$ which is 6 units jobs J_2 and J_3 have to terminate

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_1 for execution
- According to the revisited lemma job J_1 need not be preempted before time $t = 5$
- In the interval $[5, 11)$ which is 6 units jobs J_2 and J_3 have to terminate
- Considering the LO scenario the jobs combined need 7=$(5 + 2)$ units of execution

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_2 for execution

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_2 for execution
- According to the revisited lemma job J_2 can run until completion at $t = 5$

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_2 for execution
- According to the revisited lemma job J_2 can run until completion at $t = 5$
- Considering the scenario where both of the remaining jobs execute for their $C(\text{HI})$

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Assume at $t = 0$ the scheduling policy chooses J_2 for execution
- According to the revisited lemma job J_2 can run until completion at $t = 5$
- Considering the scenario where both of the remaining jobs execute for their $C(\mathrm{HI})$
- Then we need a total of $10{=}(7 + 3)$ units in the execution window $[5, 14)$ which has space for only 9 units

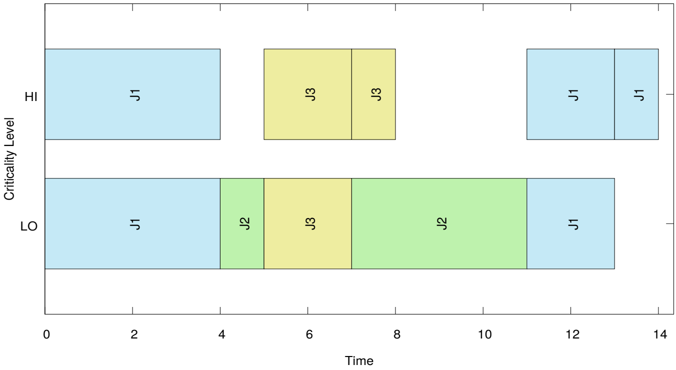
The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3

- Accoring to the 'revisited lemma' this instance is not MC-schedulable

The Counter Example

Job	A	D	χ	$C(1)$	$C(2)$
1	0	14	HI	6	7
2	0	11	LO	5	5
3	5	10	HI	2	3



Sustainability in Scheduling

- **Sustainability Property:** A scheduling policy is sustainable if an increase in the execution time of any job A while keeping all other execution times the same may not make any other job B terminate earlier [Baruah & Burns (2006)]
- Fixed-priority policy is sustainable for single- and multi-processor scheduling [Ha & Liu (1994)]
- For mixed-critical scheduling the usual sustainability definition is too restrictive as it does not take into account the possibility of a mode switch
- Many mixed-criticality scheduling policies are not sustainable under the same definition of sustainability

Sustainability in MC-Scheduling

- A scheduling policy is **MC-sustainable** if the Sustainability Property holds when we have both of the conditions below:
  - The increase of execution time of A does not lead to a change of the criticality mode in which B terminates
  - If there was no switch of criticality mode by A then the increase of execution time of A does not lead to such a switch by A
- If at least one of these two conditions is violated then B may terminate earlier

Testing Correctness

- To test the correctness of a scheduling policy one usually evaluates it for the scenario with maximal execution times for all jobs
- To perform this test a scheduling policy must be sustainable
- The adaptation of the notion of sustainability to mixed criticality raises the problem of how to adapt the policy correctness test to this new definition

Some Useful Restrictions

- For convenience we define two useful optional restrictions
- **Restriction (i)** $\chi(J_i) = \mathsf{HI} \implies C_i(\mathsf{LO}) < C_i(\mathsf{HI})$
- **Restriction (ii)** Restrict the FPM policy to have the same relative order in both priority tables ($PT_{\mathsf{LO}} \sim PT_{\mathsf{HI}}$)

MC-Sustainability of FPM

Theorem. For single-processor dual-criticality instances FPM is MC-sustainable

Theorem. If **Restriction (ii)** is satisfied then this result extends from single-processor to multi-processor instances

The sustainability property is useful for correctness testing

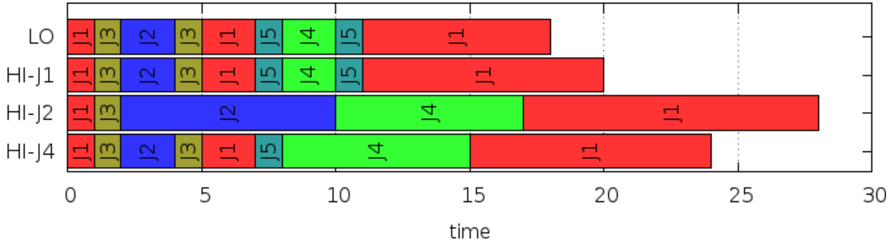
Basically Correct Policy

- **Basically Correct Policy:** An online scheduling policy is basically correct for instance J if for any basic scenario of J the policy generates a feasible schedule
- **Theorem.** If a scheduling policy is sustainable and Restriction (i) applies then the policy correctness follows immediately from its basic correctness
- Therefore correctness needs to be checked only in the basic scenarios under quite general assumptions
- However there is an exponential number of basic scenarios!
- Luckily it is enough to check only a polynomial subset of them

Canonical Algorithm for Correctness Testing

- Let S^{LO} be the schedule generated for the LO basic scenario
- Let J_h be a HI job, then we define $HI\text{-}J_h$ basic scenario such that every HI job that terminated before the termination of J_h in S^{LO} executes for LO WCET while J_h and all the other jobs execute for HI WCET
- Let $S^{HI\text{-}J_h}$ be the schedule generated for the $HI\text{-}J_h$ basic scenario
- **Theorem.** Under Restriction (i), to test correctness of an MC-sustainable policy it is enough to test it for the LO scenario and the $HI\text{-}J_h$ scenarios
- We call the test above the **Canonical Test for Correctness**

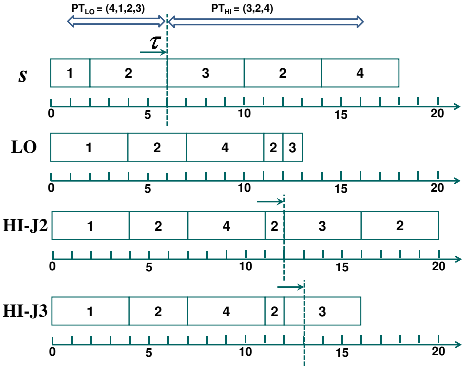
Canonical Algorithm for Correctness Testing Example



Job	A	D	χ	C(LO)	C(HI)
1	0	30	HI	10	12
2	2	10	HI	2	8
3	1	8	LO	2	2
4	8	17	HI	2	7
5	7	11	LO	2	2

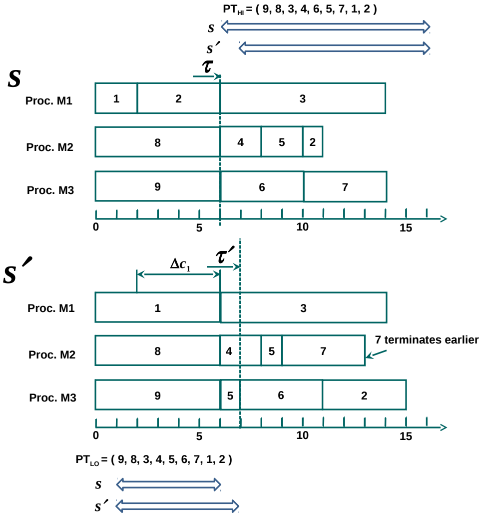
Table: Job Instance with $PT = (2, 4, 3, 5, 1)$

Restriction(i) is Necessary



Job	A	D	χ	C(LO)	C(HI)
1	0	20	LO	4	4
2	0	20	HI	4	8
3	0	20	HI	1	4
4	7	11	HI	4	4

Complication for the Multiprocessor Case



Job	A	D	χ	C(LO)	C(HI)
1	0	6	LO	6	6
2	0	14	HI	4	5
3	6	15	HI	7	8
4	6	8	HI	1	2
5	6	9	HI	1	2
6	6	11	HI	3	4
7	6	13	HI	3	4
8	0	6	LO	6	6
9	0	7	LO	6	6

Consequences for Computational Complexity

- **Theorem.** Canonical test algorithm is applicable to FPM (dual-criticality) in the following cases
  - On single processor when Restriction(i) applies
  - On multiple processors when Restriction(ii) applies
- Restriction(i) is quite general, unlike Restriction(ii), which is needed to avoid sustainability complications in multiprocessor case
- **Corollary.** Under the above restrictions FPM scheduling is in complexity class NP

Conclusion

- Whether or not the problem of deciding MC-schedulability is in NP is still an open problem
- Sustainability was adapted for mixed criticality and is important to reason about the complexity
- FPM is sustainable and in class NP under certain restrictions

References I

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., & Stougie, L. (2012). Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, *61*(8), 1140–1152.

Baruah, S., & Burns, A. (2006). Sustainable scheduling analysis. In *Real-time systems symposium, 2006. rtss'06. 27th ieee international* (pp. 159–168).

Ha, R., & Liu, J. W. (1994). Validating timing constraints in multiprocessor and distributed real-time systems. In *Distributed computing systems, 1994., proceedings of the 14th international conference on* (pp. 162–171).