Anomalies in Scheduling Control Applications and Design Complexity

Amir Aminifar[1], Enrico Bini[2]
[1]*Embedded Systems Laboratory (ESL), EPFL, Switzerland*
[2]*University of Turin, Italy*
amir.aminifar@epfl.ch, bini@di.unito.it

***Abstract*—Today, many control applications in cyber-physical systems are implemented on shared platforms. Such resource sharing may lead to complex timing behaviors and, in turn, instability of control applications. This paper highlights a number of anomalies demonstrating complex timing behaviors caused as a result of resource sharing. Such anomalous scenarios, then, lead to a dramatic increase in design complexity, if not properly considered. Here, we demonstrate that these anomalies are, in fact, very improbable. Therefore, design methodologies for these systems should mainly be devised and tuned towards the majority of cases, as opposed to anomalies, but should also be able to handle such anomalous scenarios.**

***Keywords*- control applications, resource sharing, stability analysis, time delay, latency, jitter, anomalies, design methodologies.**

I. Introduction and Related Work

Today, we are observing a shift from federated architectures to integrated architectures in automotive domain, where several applications are sharing the same platform [1]. As a result, the majority of control applications in such domains are implemented as software tasks on shared platforms (see Figure 1). This resource sharing leads to complex timing behaviors, which may jeopardize the stability of the control applications if not taken into account. This problem, known as control–scheduling co-design, has been studied in the last two decades [2]–[13].

Such complex timing behaviors, however, may lead to complexity of design methodologies. One of the main contributing factors in design complexity is the notion of anomaly resulted from these complex timing behaviors. These anomalies are simply due to the fact that important properties, e.g., the monotonicity property, do not hold anymore. Let us consider a simple abstract example, where we would like to find the maximum value of a certain parameter x, which satisfies a constraint $f(x) \leq 0$. Assuming $f(x)$ is a monotonic function with respect to parameter x, we can easily use a simple and efficient binary search algorithm and obtain the optimal solution. By checking the constraint for one value of x, we can find out if the optimum solution y satisfies $y < x$ or $y > x$. Hence efficient pruning of the search space.

Let us now consider a concrete example in the cyber-physical systems area. It is well know that a control application can provide satisfactory performance within a range of sampling periods [14]. Therefore the opportunity of optimizing control performance with respect to sampling period. It is widely believed that a controller that is allocated more computing resource (such as shorter sampling period, longer computation time, or higher priority) provides a better control quality. In this paper, instead,

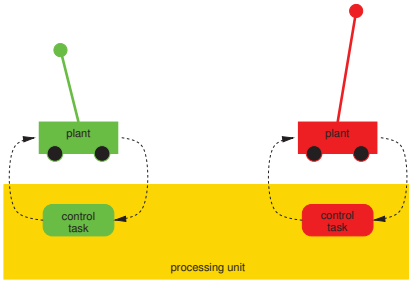

Figure 1. Resource sharing among control applications.

we demonstrate that this is actually not true. For instance, the standard quadratic control cost of a control application [14] with respect to its sampling period is shown in Figure 2. We highlight three interesting aspects of this result. First, for certain pathological sampling periods the control cost tends to infinity [15]. Secondly, allocating more resources (shorter sampling period) to a controller does not necessarily lead to a better performance (smaller control cost). Thus, ignoring this non-monotonicity can lead to unsafe and invalid design solutions. Thirdly, even though allocating more resources does not always lead to a better performance, there does exist a clear trend for control cost with respect to sampling period. That is, allocating more resources to a controller *often* leads to a better performance. Hence, ignoring this clear trend may result in a highly complex design process. This paper demonstrates that a correct design methodology should mainly target the majority of cases (i.e., exploit general trends such as monotonicity even though they do not always hold), but should also be able to handle the anomalies.

The existing optimal priority assignment, e.g., [16], and efficient sensitivity analysis, e.g., [17], in the real-time systems area are examples of exploiting the monotonicity property. However, if the monotonicity property does not hold or is ignored, often such problems cannot be solved safely and efficiently using traditional approaches. That is, in such scenarios, the majority of design methodologies (1) suffer from extreme complexity, or (2) provide invalid solutions. On the one hand, if such important properties, e.g., monotonicity, are not exploited at all, the design and optimization process becomes extremely complex. On the other hand, an approach which uses such properties without any consideration, even though they do not always hold, may produce unsafe and invalid solutions.

The anomalies in real-time task scheduling are discussed thoroughly. For example, in [18], it is shown that a shorter computation time for one task of a task chain, mapped on a multi-core platform, may lead to a longer end-to-end delay for the same task chain. Another relevant

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

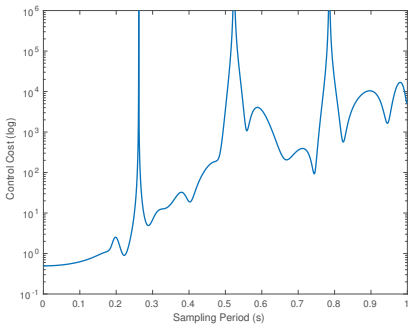

Figure 2. General increasing trend of control cost with sampling period, despite non-monotonicity.

concept is the notion of *sustainability* in the real-time scheduling area. In short, a real-time schedulability test is sustainable if any task set deemed schedulable by the test remains also schedulable with "better" parameters [19].

The anomalies in scheduling control applications are discussed in [20]. The authors show, using simple examples, that increasing the priority of a task (under the fixed-priority scheduling policy) may lead to an increased amount of jitter and, in turn, instability of the control application. Similarly, increasing the sampling period of a higher priority task may lead to instability of a lower priority task. These results, while valid, are counter-intuitive. That is, in both cases, the interference of other tasks are reduced for the task under analysis, but this leads to worse stability results.

In this paper, we discuss the anomalies in scheduling control applications and their impacts on design complexity. We show, experimentally, that these anomalies occur extremely rarely in practice. Further, we demonstrate that a correct design methodology should target the majority of cases (i.e., to exploit properties such as monotonicity), while taking the anomalies into consideration. This is illustrated for the priority assignment problem, as a case study.

II. System Model

In this paper, we assume a uniprocessor platform which is shared among n control applications. While the platform can also host other real-time tasks, for the simplicity of presentation, we only consider control applications.

A. Task Model

Given is a set of independent tasks, where each task is denoted by τ_i. The computation time (execution time), sampling period, and priority of task τ_i are denoted by c_i, h_i, and ρ_i, respectively. The computation time c_i is bounded from below by c_i^{b} and from above by c_i^{w}. The set of higher priority tasks for task τ_i is denoted by $hp(\tau_i)$. Task τ_i has higher priority than task τ_j if $\rho_i > \rho_j$. Each instance of a task is referred to as a job.

B. Plant Model

The plant associated with a control task τ_i is modeled by a continuous-time system of differential equations [14],

$$\dot{\boldsymbol{x}}_i = \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i, \tag{1}$$

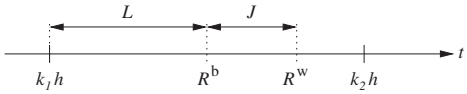

Figure 3. Graphical interpretation of the latency and worst-case response-time jitter [20].

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and the control signal, respectively. The sampling is done periodically and the control signal is updated with some delay, which depends on task scheduling.

III. Stability Analysis

In this section, we investigate the dependency between the stability of the controlled plant and the latency and jitter experienced by the control task. Therefore, in order to apply stability analysis, the values of the nominal delay (L_i) and worst-case response-time jitter (J_i) of control task τ_i should be computed. The latency L_i is defined as the constant part of the delay experienced by the control task. The jitter J_i is defined as the variation in the delay experienced by the control task. The two metrics are defined based on the worst-case and best-case response times as follows (see Figure 3),

$$\begin{aligned} L_i &= R_i^{\mathrm{b}}, \\ J_i &= R_i^{\mathrm{w}} - R_i^{\mathrm{b}}, \end{aligned} \tag{2}$$

where R_i^{w} and R_i^{b} denote the worst-case and best-case response times, respectively. In the following, we give a brief overview on computing the worst-case and best-case response times, assuming fixed-priority preemptive scheduling, implicit deadlines, and an independent task set.

Under the aforementioned assumptions, the exact worst-case response time of a task τ_i can be computed by the following equation [21],

$$R_i^{\mathrm{w}} = c_i^{\mathrm{w}} + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^{\mathrm{w}}}{h_j} \right\rceil c_j^{\mathrm{w}}. \tag{3}$$

Similarly, the exact best-case response time of a task τ_i is given by the following equation [22],

$$R_i^{\mathrm{b}} = c_i^{\mathrm{b}} + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^{\mathrm{b}}}{h_j} - 1 \right\rceil c_j^{\mathrm{b}}. \tag{4}$$

For a given controller and latency, the Jitter Margin toolbox computes the *jitter margin* (similar to the *phase margin* and *gain margin* concepts of control theory). That is, the Jitter Margin toolbox provides the stability curve that determines the maximum tolerable response-time jitter J_i, based on the experienced latency L_i.

The solid curve in Figure 4 is an example of the *stability curves* generated by the Jitter Margin toolbox, where the area below the curve is the stable area. This solid curve is generated for a DC servo process with transfer function $\frac{1000}{s^2+s}$ and a discrete-time Linear-Quadratic-Gaussian (LQG) controller, with a sampling period of 6 ms. The stability curve can safely be approximated by a linear function of the latency and worst-case response-time jitter,

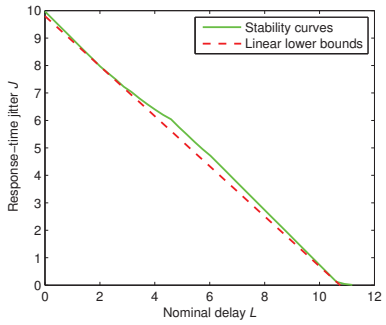

Figure 4. The stability curve generated by Jitter Margin and the linear lower bound (the area below the curve is the stable area) [20].

as shown in Figure 4. The stability condition, hence, can be formulated as,

$$L_i + a_i \cdot J_i \leq b_i, \tag{5}$$

where $a_i \geq 1$ and $b_i \geq 0$ are constant coefficients.

IV. Anomalies and Design Complexity

The anomalies in scheduling control applications are discussed in [20]. In the rest of this paper, we focus on the anomalies in the priority assignment problem for control applications, as a case study. In the case of the priority assignment problem, it is shown that increasing the priority of a task, under the fixed-priority scheduling policy, may lead to an increased amount of jitter and, in turn, instability of the control application [20]. Such anomalies, if not properly taken into account, may dramatically increase the complexity of the design methodology. Given that the monotonicity and commutativity properties [20] hold, the complexity of the priority assignment algorithm is polynomial[1] in the number of tasks [16], [23]. Ignoring the monotonicity property, however, it is possible to show that the number of cases which needs to be enumerated is exponential in the number of tasks.

Thus far, it has been discussed that anomalies may dramatically increase the complexity of the design methodology. However, as it will be shown experimentally in the next section, such anomalous scenarios occur only extremely rarely. Motivated by this, we believe that design methodologies should be devised for the majority of cases, as opposed to anomalies, but should also be able to handle the anomalous scenarios. Algorithm 1 is one such correct design methodology for the priority assignment problem.

Algorithm 1 is a backtracking algorithm for priority assignment, inspired by [24]. It identifies a task which can be assigned the lowest priority and assigns the lowest priority to this task. Then, it tries, recursively, to assign the higher priorities to the remaining set of tasks (Line 12). If there does not exist any valid priority assignment for the remaining task set, then the algorithm assigns the lowest priority to another task, which can be assigned the lowest priority. And, it again tries to assign the higher priorities to the remaining set of tasks (Line 12).

[1]The complexity of the priority assignment algorithm in [16] is quadratic in the number of tasks, if we ignore the complexity of the evaluation function.

Algorithm 1 Priority Assignment

```
1:  % T: task set;
2:  % S: remaining task set;
3:  Initialize: S = T;
4:  Initialize: ρ_i = ∞,   i = 1 ... n;
5:  Initialize: ρ = 1;
6:  function BACKTRACK(S, ρ)
7:      if S == ∅ then
8:          return % Terminate!
9:      end if
10:     for τ_i ∈ S do
11:         Set ρ_i = ρ;
12:         if L_i + a_i J_i ≤ b_i then
13:             BACKTRACK(S \ τ_i, ρ + 1);
14:         end if
15:         Set ρ_i = ∞;
16:     end for
17: end function
```

This algorithm exploits the fact that the monotonicity property holds almost all the time, by finding the task that can be assigned the lowest priority at each recursion. However, it also takes the anomalies into consideration by backtracking and trying alternative solutions.

While, in the worst case, this priority assignment algorithm has an exponential time complexity, as it will be shown in the next section, it has quadratic complexity on average. This is because the anomalies occur extremely rarely and the monotonicity property often holds.

V. Experimental Results

To support the previous discussions, in this section, we shall perform two experiments. We generate 10000 benchmarks with a set of 4–20 control applications. The plants are chosen from [4], [14]. We use the UUniFast algorithm [25] to generate a set of random control tasks for a given utilization.

First, we demonstrate that anomalies actually occur extremely rarely. To show this, we assign priorities based on the algorithm proposed in [20], modified to use the exact response times. In this way, this modified algorithm ignores the lack of monotonicity and may produce invalid solutions, for which stability cannot be guaranteed. Hence referred to as "Unsafe Quadratic". The results are summarized in Table I. In the worst case, only for 0.38% of the benchmarks, the priority assignments produced by this algorithm, Unsafe Quadratic, are not valid. This clearly demonstrates that anomalies occur extremely rarely in practice.

Table I
Percentage of invalid solutions by Unsafe Quadratic priority assignment.

Number of tasks (#)	4	8	12	16	20
Invalid solutions (%)	0.38	0.04	0.00	0.01	0.00

The second experiment shows that the time complexity of the proposed backtracking algorithm is, on average, quadratic with the number of tasks. To show this, we compare our algorithm, in terms of time complexity, with the algorithm in [20], Unsafe Quadratic, modified to use the exact response times for fair comparison. Figure 5

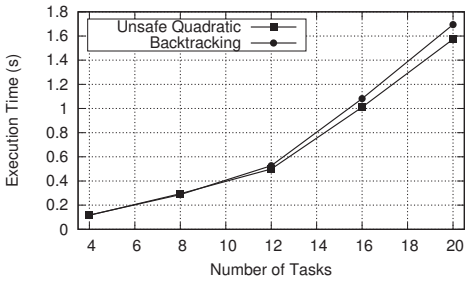

Figure 5. Execution time of our backtracking algorithm against the Unsafe Quadratic priority assignment.

shows the runtime of these algorithms on a PC with a quad-core CPU running at 3.6 GHz with 32 GB of RAM and Linux. Note that the number of all possible design solutions are 20!, which takes more than 20 years to enumerate on current GHz processors. However, Algorithm 1 finds a valid solution in less than 2 seconds.

To sum up, Algorithm 1 exploits the fact that anomalies occur very rarely and the monotonicity property almost always holds. It provides *valid* priority assignments in *reasonable time*, and in the presence of *anomalies*.

VI. Conclusions

In this paper, we considered anomalies in scheduling control applications. On the one hand, such anomalies, while can jeopardize stability, are extremely rare. On the other hand, these anomalies, if not properly considered, may lead to complex design methodologies or unstable design solutions. We demonstrate that a correct design methodology should target the majority of cases, but should also be able to handle such anomalous scenarios.

Acknowledgements

The authors would like to acknowledge Prof. Anton Cervin and Dr. Bo Lincoln for providing us with the Jitter Margin toolbox.

References

[1] M. D. Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[2] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proceedings of the* 17^{th} *IEEE Real-Time Systems Symposium*, 1996, pp. 13–21.

[3] H. Rehbinder and M. Sanfridson, "Integration of off-line scheduling and optimal control," in *Proceedings of the* 12^{th} *Euromicro Conference on Real-Time Systems*, 2000, pp. 137–143.

[4] A. Cervin, B. Lincoln, J. Eker, K. E. Årzén, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proceedings of the* 10^{th} *International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004, pp. 1–10.

[5] T. Nghiem, G. J. Pappas, R. Alur, and A. Girard, "Time-triggered implementations of dynamic controllers," in *Proceedings of the* 6^{th} *ACM & IEEE International conference on Embedded software*, 2006, pp. 2–11.

[6] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*, 2008, pp. 291–300.

[7] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task scheduling for control oriented requirements for cyber-

[8] P. Naghshtabrizi and J. P. Hespanha, "Analysis of distributed control systems with shared communication and computation resources," in *Proceedings of the 2009 American Control Conferance (ACC)*, 2009.

[9] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele, "A hybrid approach to cyber-physical systems verification," in *Proceedings of the* 49^{th} *Design Automation Conference*, 2012.

[10] G. Mancuso, E. Bini, and G. Pannocchia, "Optimal priority assignment to control tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, p. 161, 2014.

[11] A. Aminifar, P. Eles, and Z. Peng, "Jfair: A scheduling algorithm to stabilize control applications," in *Proceedings of the* 21^{st} *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015.

[12] A. Aminifar, E. Bini, P. Eles, and Z. Peng, "Analysis and desing of real-time servers for control applications," *IEEE Transactions on Computers*, 2015.

[13] Y. Xu, K. E. Årzén, A. Cervin, E. Bini, and B. Tanasa, "Exploiting job response-time information in the co-design of real-time control systems," in *IEEE* 21^{st} *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015, pp. 247–256.

[14] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems*, 3rd ed. Prentice Hall, 1997.

[15] R. Kalman, Y. Ho, and K. Narendra., *Contributions to Differential Equations*. Interscience, 1963.

[16] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Department of Computer Science, University of York, Tech. Rep. YCS 164, December 1991.

[17] R. Racu, A. Hamann, and R. Ernst, "Sensitivity analysis of complex embedded real-time systems," *Real-Time Systems*, vol. 39, pp. 31–72, 2008.

[18] R. Racu and R. Ernst, "Scheduling anomaly detection and optimization for distributed systems with preemptive task-sets," in *Proceedings of the* 12^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 325–334.

[19] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *Proceedings of the* 27^{th} *IEEE International Real-Time Systems Symposium*, 2006, pp. 159–168.

[20] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, "Stability-aware analysis and design of embedded control systems," in *Proceedings of the* 11^{th} *ACM International Conference on Embedded Software*, 2013, pp. 23:1–23:10.

[21] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[22] O. Redell and M. Sanfridson, "Exact best-case response time analysis of fixed priority scheduled tasks," in *Proceedings of the* 14^{th} *Euromicro Conference on Real-Time Systems*, 2002, pp. 165–172.

[23] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *Journal of Systems Architecture*, vol. 65, no. C, pp. 64–82, 2016.

[24] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, "Control-quality driven design of cyber-physical systems with robustness guarantees," in *Proceedings of the* 16^{th} *Conference for Design, Automation and Test in Europe (DATE)*, 2013.

[25] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.