A Position Paper on Transforming Embedded Real-Time Systems to the Cloud: Challenges and New Research Directions

Mitra Nasri and Jeroen Voeten
Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands

***Abstract*—This paper explores the design and verification of real-time cyber-physical systems transitioning from embedded platforms to the cloud. It covers opportunities, potential use cases, and challenges associated with the timing predictability of cloud-enabled real-time systems. The conclusion outlines future research directions to address these challenges.**

***Index Terms*—real-time systems, cloud, design cycle.**

I. Opportunities and Challenges

In recent years, real-time cyber-physical systems (RT-CPSes) producers have shown interest in offloading computation-intensive, soft real-time tasks to cloud platforms, delivering them as services for a fleet of systems instead of outfitting each system with powerful computational capabilities [1]–[3]. For instance, within the TRANSACT project [2], five companies across healthcare [4], maritime [5], semi-autonomous urban driving [6], electric car battery management [7], and wastewater management [8] domains have investigated the feasibility of offloading such services to edge and cloud platforms. This migration is mainly driven by considerations of cost-effectiveness, reduced in-house operational expenses (including purchasing and maintaining in-house high-performance computing platforms), and the possibility of scaling up/down computing resources on demand. However, this migration introduces new challenges in designing and verifying RT-CPSes. Before illustrating the challenges, we delve into a use case outlined in [3].

Use case (from the healthcare domain). Consider a scenario where hospitals, connected to a private cloud, request a 3D image processing service for non-intrusive surgeries (Fig. 1). Multiple requests may occur during a surgery, and several surgeries might occur simultaneously within each hospital. As outlined in [3], this service has a soft deadline (a minute), and the surgery can proceed even if the network connection to the cloud is lost (or the deadline is missed), albeit with potentially extended surgery duration. In this scenario, the surgeon is the *end user*, the computing facilities in a surgery room is the *end device*, and the provider of the 3D imaging service is the *cloud customer* (e.g., a healthcare service provider) who uses a *cloud provider* to serve the end users. The arrival time of each request to the cloud depends on the surgeon's decision, the patient's situation, and communication delays between the end device and the cloud. In an ideal scenario, each request to the 3D image processing service should be admitted on the cloud and served within its (soft) deadline.

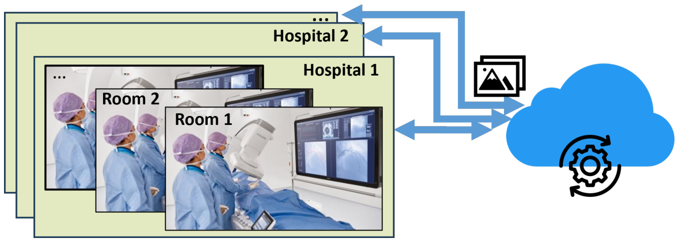

Fig. 1. During non-intrusive surgeries, surgeons send multiple requests to a cloud-based 3D imaging service [3].

Challenge 1: The arrival time of requests to the cloud is not as predictable as in an embedded system, where tasks are often periodic [9]. Conversely, network delays, delays of the virtualization platform of the cloud, and the queuing delays within the containerization technology (e.g., Kubernetes) impact the arrival time of the requests, even if they were sent periodically to the cloud. In addition, services that are offloaded are usually needed on-demand (in contrast to periodic tasks). The case study in [3] also confirms that these requests have a bursty arrival because most surgeries are planned at certain hours. If such non-determinism in the arrival pattern of requests is not taken into account, it may result in an undesirable situation where all requests are delayed far beyond their deadlines.

Challenge 2: Computing infrastructure of the cloud is shared among users. However, due to shared hardware resources such as caches and memory banks, the execution time of each request is negatively affected by other concurrently-running requests on the platform. This is known as the *co-running effect* [10,11], and can increase the execution time of a real-time task by a factor of 300 [11]. Given the non-deterministic arrival time of requests to the cloud, the number of co-running requests may vary drastically in different scenarios, resulting in unpredictable delays/jitters in the end-to-end response time of the service for the end user.

Challenge 3: Cloud platforms use virtualization technologies to provide a uniform view of the computing and memory resources for the end user. While these technologies improve flexibility and availability of resources, they also increase the communication overheads within the applications that run on the cloud. The cloud customer might request a computing node with 100 CPU cores and deploy a multi-threaded application with 100 threads to maximize parallelism. However, due to the distributed nature of these cores across different physical machines, there will be hidden communication overhead (e.g., due to data synchronization) that is hard to estimate and varies at runtime. This unpredictability introduces two challenges: (i) an increase in the *worst-case execution time* (WCET) of requests, and (ii) increased difficulty in obtaining a reliable bound on WCET using *measurement-based timing analysis* (MBTA) techniques used in industrial real-time systems [9,10].

II. Future Research Directions

Fig. 2 shows a typical design cycle of real-time systems, where the initial implementation is used to extract a timing model for the workload (i.e., the arrival model and WCET of each task). More than 50% of industrial real-time systems use in-house or third-party tools to measure WCET [9]. The workload model, along with platform and scheduling policy models, is used in the response-time analysis stage. This step employs theoretically sound techniques (e.g., fixed-point iteration [12]–[14], schedule-abstraction [15]–[19], and timed automata [20]) to determine the *worst-case response time* (WCRT) for each task on the platform. If WCRT exceeds the deadline, adjustments may be necessary, such as redesigning or reconfiguring the system (e.g., altering task-to-core mapping or priority assignment). Alternatively, the system might employ runtime techniques like watchdog timers, cache partitioning, or reservation servers (e.g., constant bandwidth servers [21]) to enforce timing predictability and temporal isolation.

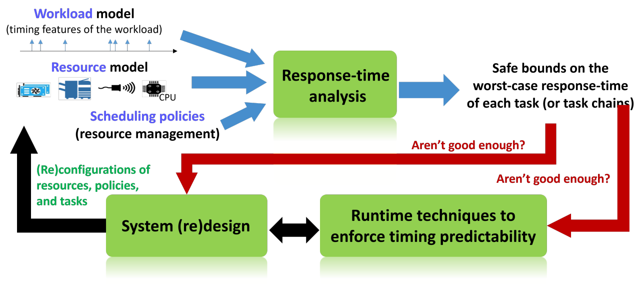
Workload model
(timing features of the workload)
Resource model
CPU
Scheduling policies
(resource management)
Response-time analysis
Safe bounds on the worst-case response-time of each task (or task chains)
Aren't good enough?
Aren't good enough?
(Re)configurations of resources, policies, and tasks
System (re)design
Runtime techniques to enforce timing predictability

Fig. 2. A typical design cycle for real-time systems involves initial implementation, derivation of timing models, and a response-time analysis. If the outcome is not good enough, a redesign is needed and the cycle repeats.

Deriving high-fidelity timing models to characterize the workload on the cloud. Addressing Challenges 1 to 3 involves establishing a high-fidelity timing model for the workload, including an accurate arrival model and an upper bound on the worst-case execution time (WCET) of requests. This is essential for ensuring the timing requirements of cloud-enabled real-time systems.

Arrival curves are one of the most flexible models to characterize the bursty nature of cloud requests, allowing to model the minimum and maximum inter-arrival times between consecutive requests of a user. Additionally, these curves can be combined to represent cumulative workloads of multiple users [22]. However, sound and automated techniques still need to be developed to obtain fateful arrival curves.

Working with partially fateful timing models. As highlighted in Challenge 2, the co-running effect can significantly inflate the execution time of a request, potentially by a factor of 300. Relying solely on meeting deadlines, even with an execution time that is 300 times larger than the average, may result in an inefficient design that could force the cloud customers to pay for resources that remain idle most of the time. Moreover, considering the vast search space for WCET or the arrival pattern of requests, obtaining fateful arrival curves or safe upper bounds on WCET may not be attainable.

This opens doors to response-time analysis for partially fateful models, including (i) probabilistic response-time analysis [23,24] or (ii) Vestal model for mixed-criticality systems [25], where the execution time is described by multiple thresholds, each corresponding to a criticality level. Crossing each threshold may change the criticality level of the system, triggering a mechanism to either stop or degrade its service to lower-criticality functionalities in the system. Despite their potentials, neither approach (i) nor (ii) has been adapted to cloud platforms. In particular, most probabilistic response-time analyses assume that the execution time of a task is independent of the execution time of another task, meaning that execution times are *independent and identically distributed*. However, this assumption does not hold due to Challenge 2.

The need for runtime techniques for timing predictability. In light of Challenges 2 and 3, a promising approach is to leverage a combination of monitoring techniques, admission tests, non-preemptive execution, and dynamic resource management techniques. These techniques can help in (i) estimating the current and future workload, (ii) controlling the input workload, (iii) minimizing potential interferences due to micro-architectural properties or task migrations, and (iv) adjusting resources (and priorities) dynamically to meet the timing requirements of admitted requests.

The need for more flexible timing requirements. In some cases, Challenge 1 may involve establishing a service-level agreement between the cloud customer (healthcare service provider) and users (e.g., hospitals) to ensure a minimum service level. The *weakly-hard timing constraints*, e.g., the (m, k) model is one of the most flexible ways to represent such agreements. Under the (m, k) model, the cloud customer commits serving a minimum of m requests out of every k consecutive requests from a specific (super)user. Multiple (m, k) constraints can be stacked to account for both short-term and long-term behavior. For instance, $(900, 1000) \wedge (2, 5)$ implies ensuring at least 900 out of every 1000 requests are served and limiting the consecutive missed requests to no more than 3 (= 5 - 2).

Response-time analysis must consider the runtime techniques used by a cloud customer. This would mean they not only need to accommodate dynamic resource management policies but also work with arrival curves and weakly-hard timing constraints on a platform that likely uses other isolation techniques such as reservation-based scheduling.

Acknowledgment

This work was supported by the EU ECSEL project TRANSACT (grant no. 101007260).

References

[1] "Software-Defined Vehicles – A Forthcoming Industrial Evolution," https://www2.deloitte.com/cn/en/pages/consumer-business/articles/software-defined-cars-industrial-revolution-on-the-arrow.html, 2015.

[2] "TRANSACT Project," https://transact-ecsel.eu/, 2022.

[3] T. Hendriks, B. Akesson, J. Voeten, M. Hendriks, J. C. Parada, M. García-Gordillo, S. Sáez, and J. J. Valls, "Thirteen concepts to play it safe with the cloud," *IEEE International Systems Conference (SysCon)*, 2023.

[4] "Surgical Planning Everywhere," https://transact-ecsel.eu/2022/04/edge-cloud-based-clinical-applications-platform/, 2024.

[5] "Navigational Safety at Sea," https://transact-ecsel.eu/2022/03/critical-maritime-decision-support/, 2024.

[6] "Revolutionary Urban Public Transport," https://transact-ecsel.eu/2022/03/remote-operations-of-autonomous-vehicles-for-navigating-in-urban-context/, 2024.

[7] "Energy Efficient Electric Vehicles," https://transact-ecsel.eu/2022/03/cloud-featured-battery-management-systems/, 2024.

[8] "Safe and Efficient Wastewater Recycling," https://transact-ecsel.eu/2022/05/critical-wastewater-treatment-decision-support/, 2024.

[9] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "A comprehensive survey of industry practice in real-time systems," *Real-Time Systems Journal*, pp. 1–41, 2021.

[10] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Compututing Systems*, vol. 7, no. 3, 2008.

[11] M. Bechtel and H. Yun, "Denial-of-service attacks on shared cache in multicore: Analysis and prevention," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 357–367.

[12] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[13] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *Real-Time Systems Symposium (RTSS)*. IEEE, 2007, pp. 119–128.

[14] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) schedulability analysis : refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[15] M. Nasri and B. B. Brandenburg, "An exact and sustainable analysis of non-preemptive scheduling," in *Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 12–23.

[16] M. Nasri, G. Nelissen, and B. B. Brandenburg, "A response-time analysis for non-preemptive job sets under global scheduling," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2018, pp. 9–1.

[17] ——, "Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2019, pp. 21–1.

[18] S. Ranjha, G. Nelissen, and M. Nasri, "Partial-order reduction for schedule-abstraction-based response-time analyses of non-preemptive tasks," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 121–132.

[19] S. Ranjha, P. Gohari, G. Nelissen, and M. Nasri, "Partial-order reduction in reachability-based response-time analyses of limited-preemptive DAG tasks," *Real-Time Systems*, pp. 1–55, 2023.

[20] B. Yalcinkaya, M. Nasri, and B. B. Brandenburg, "An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks," in *DATE*, 2019, pp. 1228–1233.

[21] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, 3rd ed. Springer Science & Business Media, 2011.

[22] M. Moy and K. Altisen, "Arrival curves for real-time calculus: The causality problem and its solutions," in *Tools and Algorithms for the Construction and Analysis of Systems*, J. Esparza and R. Majumdar, Eds. Springer Berlin Heidelberg, 2010, pp. 358–372.

[23] S. Bozhko, G. von der Brüggen, and B. B. Brandenburg, "Monte carlo response-time analysis," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 342–355.

[24] S. Adyanthaya, M. Geilen, T. Basten, J. Voeten, and R. R. H. Schiffelers, "Iterative robust multiprocessor scheduling," in *International Conference on Real Time Networks and Systems (RTNS)*. ACM, 2015, pp. 23–32.

[25] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239–243.