Online latency monitoring of time-sensitive event chains in safety-critical applications

Jonas Peeck, Johannes Schlatow and Rolf Ernst
Institute of Computer and Network Engineering
TU Braunschweig
Braunschweig, Germany
{peeck,schlatow,ernst}@ida.ing.tu-bs.de

***Abstract*—Highly-automated driving involves chains of perception, decision, and control functions. These functions involve data-intensive algorithms that motivate the use of a data-centric middleware and a service-oriented architecture. As an example we use the open-source project Autoware.Auto. The function chains define a safety-critical automated control task with weakly-hard real-time constraints. However, providing the required assurance by formal analysis is challenged by the complex hardware/software structure of these systems and their dynamics. We propose an approach that combines measurement, suitable distribution of deadline segments, and application-level online monitoring that serves to supervise the execution of service-oriented software systems with multiple function chains and weakly-hard real-time constraints. We use DDS as middleware and apply it to an Autoware.Auto use case.**

I. Introduction

In a general perspective, highly-automated driving requires software-intensive cyber-physical systems (CPSs) with sophisticated perception capabilities as input to vehicle motion control. For the automotive domain, the AUTOSAR Adaptive Platform (AP) provides the major standard for implementation of these systems. In particular, it specifies [1] the use of a data-centric middleware, namely the Data Distribution Service (DDS) [2] or SOME/IP [3] to exchange data between software modules following a service-oriented architecture. Perception is time-sensitive including, e.g., processing of lidar data by the use of multiple threads/processes that build a software pipeline. This way, the required throughput (i.e. frame rate) can be reached if all processes achieve a certain response time. Yet, in the context of environment perception, the end-to-end (e2e) latency of the entire function chain is at least equally important as it determines the freshness of the data on which trajectory planning is performed. This e2e latency not only includes the response times of the involved processes but also non-negligible communication delays, resulting from scheduling effects, library code, kernel execution and network interference. More generally, a function chain can thus be perceived as a chain of middleware communication events.

As soon as these time-sensitive event chains are related to highly-automated or even autonomous driving, they become safety critical. This also means it must be prevented that obsolete data is used for automated decision-making. Providing e2e latency guarantees in high-performance architectures is still an unsolved challenge. These architectures, software libraries, middleware, and data-dependent algorithms used in perception systems are not easily amenable to hard real-time guarantees, unless we accept major performance losses for pessimistic bounds that degrade perception quality thereby increasing the risk of functional failures and hazards. Fortunately, the algorithms fall in the category of weakly-hard real-time systems, i.e. they tolerate a limited number of deadline misses (i.e. violations of a latency requirement). A weakly-hard latency requirement (m, k) denotes that the latency constraint must not be exceeded more than m times out of k consecutive executions [4]. Weakly-hard real-time systems and related formal analysis methods are meanwhile well investigated [5]. However, exploitation of (m, k) constraints for performance improvement is difficult and not only suffers from the complex timing behaviour of high-performance architectures, but the impact of an (m, k) violation likely depends on the driving context. A possible way out is to involve the application in handling these violations. There are, however, many obstacles in observing, detecting and classifying timing violations in complex architectures and reflecting them to the application level with minor performance overhead.

We, therefore, present a decentralized monitoring concept supporting low-overhead supervision of event-chain latencies at runtime. It establishes an independent safety mechanism for detection and reaction to latency violations in real time. The paper is structured as follows: We concretize the problem statement in Section II based on a use case from Autoware.Auto. In Section III, we briefly review related work on monitoring approaches before we present our monitoring concept in Section IV. Finally, we conclude in Section V.

II. Use case

With its high utilization and integration in industry standards, DDS will be an important constituent of future data-centric CPS. A popular platform that exploits DDS is the Robot Operating System (ROS) 2 framework. Because of the similarity in environment perception functionalities between robotics and highly-automated automotive vehicles, ROS 2 has been adopted by the Autoware foundation in their popular open-source project Autoware.Auto. In this paper, we focus on the lidar-based environment-perception stack from Autoware.Auto that builds our use case. This use case is depicted in Fig. 1, which shows the distribution and communication of the

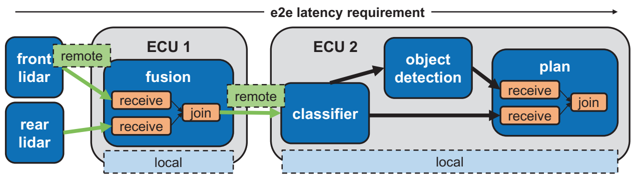

Fig. 1: Autoware.Auto perception stack example.

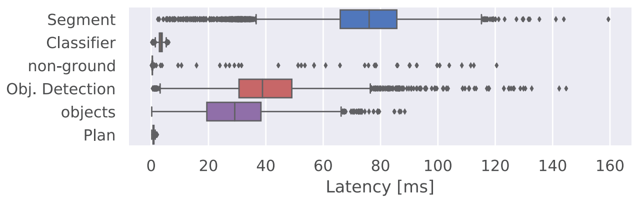

Fig. 2: Measured individual and segment latencies on ECU2 running ROS2 with Linux on an Intel Core i5-3210M.

involved ROS nodes. In this example, a ROS node (shown as blue boxes) corresponds to a separate single-threaded process that subscribes to topics that are made available via the DDS middleware and publishes to topics on its own. In the remainder of this paper, we therefore denote ROS nodes as services. A process is activated whenever a message for one of the subscribed topics arrived, in which case the corresponding activities (also known as callbacks) of the process are executed (shown as encapsulated orange boxes). DDS transparently transfers the communicated data between the ROS nodes executing on the same or on different Electronic Control Units (ECUs). From a latency perspective, however, we need to distinguish communication in a local domain (on the same ECU, depicted as black arrows) from remote communication over Ethernet (depicted as green arrows). The use case depicted in Fig. 1 comprises a front lidar and a rear lidar that periodically send point-cloud data as separate topics to the fusion service on ECU 1. Here, we assume that the lidar services are provided as third-party hardware modules. The fusion service joins the data of both lidars based on their timestamps and sends a combined point-cloud data to the classifier service on ECU 2, which classifies the points and publishes the ground points and non-ground points as separate topics. The object-detection service receives the non-ground points and performs a clustering algorithm in order to publish the detected objects. Based on the detected objects and the ground points, the plan service decides on the trajectory that the vehicle shall follow.

The vehicle's safety depends on the latency of this processing chain, since trajectories must not be used if their sensor data input is outdated. While a rare violation of the corresponding end-to-end deadline can be accepted, consecutive deadline misses may require fallback to a safety layer. Hence, there is a weakly-hard e2e latency requirement. More precisely, the latency from publishing the front/rear lidar data to publishing the planned trajectory shall be below a certain value in m out of k consecutive activations. Any violation is thus considered a fault that must be handled appropriately for safety reasons. Note, that this requirement must hold for any path (i.e. event chain) in this processing chain.

To illustrate the relevance of latency monitoring, we traced this use case with LTTng [6] and measured the individual processing and communication latencies of the services on ECU 2 as well as the combined latency for the chain segment from the start of the classifier to the end of the plan service. As automotive high-performance architectures base on modern x86 or ARM CPUs with POSIX-compatible OSs, we used a standard consumer CPU with Linux. The periodicity of the chain is approx. 120 ms. Figure 2 depicts the results as Tukey boxplots and shows that the communication latencies (non-ground, objects) are significant. Furthermore, the longest individual latency occurs for the object detection, which therefore determines the frame rate. The segment latency is a measure for the data age that is accumulated on ECU 2. Interestingly, it is much smaller than the sum of the individual latencies, which motivates the need to monitor across different processes of a chain. While this evaluation is straightforwardly applied in terms of an offline analysis, there is no existing solution for online monitoring of these latencies.

Monitoring is a reasonable approach to detect these violations as it bases on independent system entities (monitors) that continuously observe critical properties. However, existing and straightforward approaches are insufficient for monitoring the latency of event chains.

III. Related Work

Online monitoring, also known as runtime monitoring or runtime verification is a long known concept for providing assurance in critical domains. It bases on the continuous observation of critical properties by independent system entities (monitors). Early approaches aim to validate the correctness of execution of real-time systems at runtime [7] or target to protect critical system functions by early violation detection and corrective action [8]. Since then, several frameworks evolved around runtime monitoring concepts like [9] or [10]. Another approach for isolating tasks or groups of tasks is to monitor the arrival patterns of arbitrary tasks as well as their arrival workload at runtime [11], [12]. An overview of different monitoring techniques for safety-critical multicore systems is provided in [13]. As automotive systems are getting much more complex while safety requirements due to upcoming autonomous driving increase, [14] emphasises the importance of monitoring Quality of Service (QoS) parameters like deadlines.

However, existing monitoring methods are restricted to processing delays or communication delays only and therefore insufficient for monitoring the latency of event chains. For instance, AUTOSAR AP supports a checkpointing approach [15] that can be used to observe the time between events of the same process but which neglects communication latencies between processes. On the other hand, DDS employs a QoS mechanism to supervise message deadlines (i.e. their inter-arrival time) but which neglects the processing delays. Yet, combining both mechanisms would lead to significant overestimation of chain latencies as illustrated in Section II.

In addition to the mentioned works, system behaviour can be analysed by many existing tracing frameworks [6], [16], [17]. However, these produce a high data volume and are thus not directly applicable to online monitoring, which shall be able to timely react to latency violations to provide assurance.

IV. Monitoring Concept

In the above example, the most obvious approach is to monitor the rate of published messages at the plan service. While this is suitable to detect lost or delayed messages, it does not give an indication about the age of the lidar data from which the trajectory was calculated. Instead, for any time-sensitive event chain c, we want to ensure that it never exceeds a certain e2e latency budget B^c_{e2e} more than m times within a window of k consecutive activations. This could be achieved by a centralised monitor that receives copies of the published messages (or only their timestamps). Yet, due to the distributed processing, this would require additional network communication that may interfere with the normal operation of the system. Another possibility is to use local monitors to observe the time from activation (resume) to completion (suspend) of individual processes (i.e. their response time). While this reduces the problem to the supervision of local deadlines that can be efficiently implemented within the individual processes, it neglects the communication latencies. Our approach is based on the idea to establish a monitoring mechanism that is able to observe and relate communication events from different processes on the same ECU and to combine this with a monitoring mechanism for network latencies. The result is a segmentation of the e2e latency requirement into local and remote segments (cf. Fig. 1). Local segments start with a receive event and end with a publication event. For remote segments it is vice versa. This approach detects when an event has not occurred until a certain deadline thereby enables to perform recovery or degradation. The challenges with this approach lie within the determination of segment deadlines, the efficient implementation of the monitoring mechanisms for local and remote segments, and the low-latency fault handling in case of detected violations.

In the remainder of this section, we present the essential details of our monitoring concept; further details are available in our technical report [18]. First, we formulate and constrain a valid segmentation of an event chain. In a second step, we point out how to handle deadline violations of single segments in order to comply to weakly hard (m, k)-constraints of a chain. Thereafter, we briefly describe the basic concepts for local and remote segment monitoring.

A. Segmentation of e2e event chains

An event chain c as depicted in Fig. 1, is a sequence S_c of local and remote segments s_i. To allow independent monitoring mechanisms, we thus distribute the chain's budget by defining segment deadlines d^{s_i}. Monitoring detects deadline misses and either performs a recovery (i.e. the next segment can execute normally) or a propagation of the deadline miss to the next segment. The monitoring actions are realised by temporal exceptions as explained in Section IV-B. For determining the actual d^{s_i}, we formulate the following constraint satisfaction problem that bases on a trace of latency measurements. Here, $l'^{s_i}_n$ denotes the latency of segment s_i (including exception handling) at position n in the trace.

$$\begin{array}{lll} \text{find} & d^{s_i} \in \mathbb{N} & \forall s_i \in S_c \quad (1) \\ \text{subject to} & B^c_{e2e} \geq \sum_{s_i \in S_c} d^{s_i} & (2) \\ & m \geq \max_n M_i(n) & \forall s_i \in S_c \quad (3) \\ \text{with} & m_i(n) = |\{j | n \leq j \leq n+k \wedge l'^{s_i}_j > d^{s_i}\}| & (4) \\ & M_i(n) = m_i(n) + \sum_{l=0}^{n-1} p_l \cdot m_l(n) & (5) \end{array}$$

Eq. (2) formalises that the sum of segment deadlines must not exceed the total e2e latency budget. Eq. (3) ensures that at any position n in the trace, there are not more than m misses for every segment including unrecoverable misses of preceding segments. Here, Eq. (4) calculates the number of segment deadline misses that occur between the n-th and $n+k$-th activation of the segment. Eq. (5) takes propagation of deadline misses from preceding segments into account where p_l is the propagation factor of the segment s_l that can assume 0 (in case of perfect recovery) or 1 (in case of propagation). For $p_l = 0$, the constraint satisfaction problem is split into several single-variable problems for each segment. For $p_l = 1$, we refer to heuristic methods or integer linear programming (ILP), which is out of the scope of this paper.

B. Temporal exceptions

The basic idea of latency monitoring is to continuously observe communication events at runtime and to raise *temporal exceptions* whenever a segment does not finish in time. The segment's exception handler can either recover from a deadline violation and still provide usable data, or it can propagate the violation to the next segment, e.g., by cancelling the next publication or receive event. The propagation of all unrecoverable violations is essential, as it allows us to use the (m, k)-constraint from B^c_{e2e} also for the individual segment deadlines d^{s_i}, thus enabling a decentralized monitoring approach. A chain activation thus fails if any of it's segments raises an unrecoverable temporal exception.

In order to enable a recovery, violations must be detected before d^{s_i} and the latency of the exception handler itself must be bounded. The latter can be achieved by executing the exception handling on the highest scheduling priorities. We therefore split d^{s_i} into a monitored deadline $d^{s_i}_{\text{mon}}$ and the maximum latency of the exception handling d_{ex}: $d^{s_i} = d^{s_i}_{\text{mon}} + d_{\text{ex}}$. In case of a violation, an exception is raised directly after $d^{s_i}_{\text{mon}}$ and completed before d^{s_i}. A single violation of a segment's deadline $d^{s_i}_{\text{mon}}$ can have multiple reasons, like a data-dependent execution time, sporadic system overload, memory interference or even a deadlock. However, all those reasons are transparent to the monitor, which aims to keep safe operation up. In case of a deadline violation an exception is raised, which can then be handled by the application itself or a system-level entity to perform further diagnostics or take appropriate countermeasures. Using exceptions to evoke application-level actions is justified, because only at that level it can be decided if a particular temporal exception corresponds to a fault. This approach avoids false positive failure diagnosis as would result from current low-level monitoring.

C. Monitoring of local segments

For the latency monitoring of local segments, we leverage the fact that the start event and the end event of a segment occur on the same processing resource. In middleware concepts like DDS, events correspond to communication events such as the reception or publication of a message. These events typically occur in different processes such that timestamps of the start events must be made available to the process in which the end event will occur. For this purpose, additional inter-process communication is required that should be faster than the communication primitives provided by the middleware. We can therefore leverage shared memory and inter-process notifications such as semaphores, while a timeout is deployed in order to trigger an exception after the violation of the segments deadline. Our implementation showed a maximum overhead of $86\,\mu s$ for a communication event, which is much smaller than the overall chain latency. In case the deadline is violated, the exception handler takes appropriate measures. Such a monitoring mechanism is independent of the particular operating system or middleware implementation.

D. Monitoring of remote segments

In contrast to the monitoring of local segments, the start and end events of remote segments occur on different processing resources. A remote segment ends with a receive event so that a monitor has to be implemented at the receiver side. As a consequence, we have to make assumptions on the time of the start event in order to initialize a monitoring timeout for detecting deadline violations as its concrete value is unknown. A sophisticating approach can benefit from a periodic communication together with time synchronization of control units, which is provided via Precision Time Protocol (PTP) [19] in modern cars. With known period P_{s_i}, the start event timestamp $t^{s_i}_{\text{st,n}}$ has to be transmitted together with the data and can be used to program the timer for the reception of the $n+i$-th end event: $t^{s_i}_{\text{st,n}} = t^{s_i}_{\text{st,n-1}} + (i+1)P_{s_i} + d^{s_i}_{\text{mon}}$. Note, $d^{s_i}_{\text{mon}}$ includes the message's publication jitter J_{s_i} as well as the synchronization error ϵ between the ECUs, which can both be determined from traces. Deviations from J_{s_i} and ϵ assumptions do not lead to undetected latency violations: Imagine an activation occurs too late, this would result in a closer deadline for the corresponding transmission, as we programmed the timer based on the previous start event. On the other hand, an unexpected early activation would result in the case, that the transmission could exceed the segment's budget without detection. However, this can only be the case, if the previous segments along the chain finished earlier and thereby left slack to be used by the remote segment.

V. Conclusion

Analytical methods are too pessimistic or insufficient for e2e latency guarantees on high-performance platforms running service-oriented middleware. In such cases, online monitoring seems a possible solution for e2e event-chain supervision and control of safety layer transition. Yet, existing solutions are limited to monitoring individual processing or communication delays and, thus, neither capture e2e latencies accurately nor respect weakly-hard requirements of practical applications. This paper presents monitoring of arbitrary communication events across process boundaries thereby capturing delays and requirements from all system layers. It is independent from the application logic and, hence, suitable to support safety mechanism. We demonstrated its application to an important use case running DDS on the popular platform Autoware.Auto.

References

[1] "AUTOSAR Specification of Communication Management for Adaptive Platform R19-11."

[2] Object Management Group (OMG), "Data Distribution Service, Version 1.4," OMG Document Number formal/2015-04-10, March 2015.

[3] "AUTOSAR SOME/IP Protocol Specification R19-11."

[4] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[5] Z. A. H. Hammadeh, S. Quinton, R. Henia, L. Rioux, and R. Ernst, "Bounding Deadline Misses in Weakly-Hard Real-Time Systems with Task Dependencies," in *Design Automation and Test in Europe*, Lausanne, Switzerland, Mar. 2017.

[6] M. Desnoyers and M. Dagenais, "Lttng: Tracing across execution layers, from the hypervisor to user-space," in *Linux symposium*, vol. 101, 2008.

[7] M. Kim, M. Viswanathan, H. Ben-Abdallah, S. Kannan, I. Lee, and O. Sokolsky, "Formally specified monitoring of temporal properties," in *Euromicro Conference on Real-Time Systems*, 1999.

[8] F. Jahanian, R. Rajkumar, and S. C. Raju, "Runtime monitoring of timing constraints in distributed real-time systems," *Real-Time Systems*, vol. 7, no. 3, pp. 247–273, 1994.

[9] S. Kannan, M. Kim, I. Lee, O. Sokolsky, and M. Viswanathan, "A retrospective look at the monitoring and checking (mac) framework," in *International Conference on Runtime Verification*. Springer, 2019.

[10] L. Pike, N. Wegmann, S. Niller, and A. Goodloe, "Copilot: monitoring embedded systems," *Innovations in Systems and Software Engineering*, vol. 9, no. 4, pp. 235–255, 2013.

[11] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst, "Monitoring arbitrary activation patterns in real-time systems," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2012.

[12] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst, "Monitoring of workload arrival functions for mixed-criticality systems," in *Proc. of Real-Time Systems Symposium (RTSS)*, Dec. 2013.

[13] S. Tobuschat, A. Kostrzewa, F. K. Bapp, and C. Dropmann, "Online monitoring for safety-critical multicore systems," *it - Information Technology*, 2017.

[14] J. Ahluwalia, I. Krger, W. Phillips, and M. Meisinger, "Model-based run-time monitoring of end-to-end deadlines," in *ACM International Conference on Embedded Software (EMSOFT)*, Jan. 2005.

[15] "AUTOSAR Specification of Platform Health Management for Adaptive Platform R19-11."

[16] M. Maggio, J. Lelli, and E. Bini, "rt-muse: measuring real-time characteristics of execution platforms," *Real-Time Systems*, vol. 53, no. 6, pp. 857–885, Nov. 2017.

[17] M. Garca-Gordillo, J. J. Valls, and S. Sez, "Heterogeneous runtime monitoring for real-time systems with art2kitekt," in *International Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[18] J. Peeck, J. Schlatow, and R. Ernst, "Online latency monitoring of time-sensitive event chains in ROS2," TU Braunschweig, Tech. Rep., Jan. 2021.

[19] "IEEE 1588-2008 - Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," IEEE, Standard, Jul. 2008.