Exploring Partitioned and Semi-partitioned Callback Scheduling on ROS 2 Multi-threaded Executors

Hoora Sobhani Daniel Enright Tejas Milind Deshpande Hyoseung Kim
University of California, Riverside
{ hsobh002, denri006, tdesh006, hyoseung }@ucr.edu

Abstract

In recent studies aimed at enhancing the analyzability and real-time performance of ROS 2, there has been insufficient emphasis on the importance of different scheduling options, including global, partitioned, and semi-partitioned approaches, particularly when multiple CPU cores are involved. In this work, we enabled the partitioned and semi-partitioned scheduling for ROS 2 multi-threaded executors and discussed the opportunities and the potential issues associated with it.

I. Introduction

The Robot Operating System (ROS) is a versatile middleware platform widely used for developing robotic systems and applications. The increasing need for robust, real-time capabilities in modern robotic applications such as autonomous drones, fully autonomous vehicles, and mobile robots has driven the development of ROS 2. This successor to ROS offers a modular architecture and an extensive set of tools and libraries, forming a flexible framework that enhances performance, increases reliability, and provides real-time system support. Numerous recent studies from the real-time community underscore the importance of ROS 2 in enabling serious, safety-critical industrial applications, highlighting its pivotal role in modern robotic system development [1]–[12].

ROS 2 introduces key abstractions such as executors, which are OS-level processes, and callbacks, which are the smallest schedulable units within these processes. One notable feature of ROS 2 is the support for both single-threaded and multi-threaded executors. Multi-threaded executors enable the parallel execution of callbacks across multiple threads and is currently utilizing a global scheduling approach for these callbacks. However, there is an ongoing debate in the real-time systems community about the real-time performance of partitioned scheduling, semi-partitioned scheduling, and global scheduling. While global scheduling offers the most flexibility and resource-sharing capabilities, exploring other scheduling approaches may provide additional benefits. Specifically, partitioned scheduling offers strong predictability through temporal isolation of tasks, while semi-partitioned scheduling strikes a balance between resource sharing, isolation, and flexibility.

We intend to enable partitioned and semi-partitioned callback scheduling approaches for ROS 2 multi-threaded executors and compare their performance with the global scheduling approach.

II. Background & related work

A. ROS 2 Architecture

ROS 2 middleware is built on top of the operating system and provides a comprehensive suite of tools and libraries. This includes the client library (rcl), which comprises language-specific APIs such as rclcpp for C++ and rclpy for Python, and the middleware libraries (rmw), which handle communication between nodes. The rmw libraries support publisher-subscriber interfaces, services, and actions, facilitating both inter-process and intra-process communication. Figure 1 illustrates the architecture and application model of ROS 2 [8].

A ROS 2 application is organized around the abstraction of nodes, each node comprising a set of callbacks. These callbacks can be one of five types: timer, subscription, service, client, or waitable. Timer callbacks are triggered periodically, while the others are event-triggered. Once a node is assigned to an executor, all its callbacks are subsequently managed by that executor. Each executor, whether single-threaded or multi-threaded, maintains an exclusive *ReadySet* of pending callbacks and selects a callback for execution based on its priority [1], [2], [8]. In ROS 2, the priority of a callback within an executor is determined by its type. Timer callbacks always have the highest priority. Among non-timer callbacks, the priority order is predefined, typically following this sequence: subscription, service, client, and waitable. For single-threaded executors, the executor will execute a single callback from the ReadySet on the CPU while multi-threaded executors may dynamically execute multiple callbacks simultaneously across as many threads as it is allocated. The current practice for executing ready callbacks on threads in multi-threaded executors is through a work-conserving global scheduling approach.

B. Global and Partitioned Scheduling Approaches

Global and partitioned scheduling are essential approaches in multi-processor real-time systems, each with distinct advantages and disadvantages. Global scheduling offers flexibility and maximizes resource utilization by enabling tasks to run across multiple processors. However, it can increase overhead due to the complexity of managing tasks and resource contention across processors, making it less ideal for hard real-time systems that require consistent predictability and strict timing.

In contrast, partitioned scheduling enhances predictability and reduces overhead by statically assigning tasks to specific processors. This approach simplifies the scheduling process and minimizes interference from other processors, although it might not utilize processor resources as effectively as global scheduling [13].

Semi-partitioned scheduling serves as a compromise, allowing constrained task migration to enhance resource utilization while maintaining more predictability than global scheduling. This method seeks to increase resource utilization while simultaneously satisfying the need for reliable task execution [14].

C. Scheduling Callbacks within ROS 2 Executors

Scheduling approaches in the real-time domain are overwhelmingly well-documented, yet the exploration of scheduling methodologies within ROS 2, especially in multi-core environments, remains comparatively sparse. Previous research has primarily focused on response-time analysis and enhancements in the overall real-time capabilities within ROS 2, as seen in several pioneering studies [1]–[9]. These papers typically adopt one of two approaches when involving scheduling callbacks on multiple CPU cores: enforcing partitioned scheduling across multiple single-threaded executors [3], [5] or employing the default global scheduling behavior within a multi-threaded executor [7], [8]. Despite these contributions, to the best of our knowledge, there are no studies addressing the integration of partitioned and semi-partitioned callback scheduling strategies for multi-threaded executors in ROS 2.

III. Partitioned and Semi-partitioned Scheduling within the ROS 2 Multi-threaded Executor

To enable partitioned and semi-partitioned scheduling in ROS 2 multi-threaded executors, we have introduced a new thread affinity API to the *rclcpp* library. This API allows developers to specify which specific threads within the multi-threaded executor a callback may execute on, thus offering precise control over task scheduling across the executor's threads. The scope of this API extends beyond the multi-threaded executor class to modifications in callback-related classes, supporting both the default and priority-driven executors for all types of ROS 2 callbacks. [1]

This enhancement enables developers to choose between global, partitioned, and semi-partitioned scheduling strategies by designating specific threads for certain callbacks. Additionally, it facilitates the reservation of execution bandwidth for high-priority callbacks, thereby improving predictability for hard real-time systems. By default, each callback can execute on any thread unless developers explicitly restrict its thread affinity. This API also brings enhancements to ROS 2's multi-threaded executors by offering greater determinism compared to global scheduling and improving resource utilization over traditional partitioned scheduling. However, these changes introduce some complexities and overhead. For instance, in the current ReadySet implementation, there is no guarantee that an idle thread attempting to retrieve ready callbacks will be able to execute any of the callbacks in the set. This can result in some idle executor threads while some callbacks might be deprived of compute resources. A significant challenge is the need for advanced callback-to-thread allocation algorithms within the executor that can consider information about callback utilization and execution times. Given ROS 2's reliance on publisher-subscriber communication and event-triggered callbacks, determining the most effective allocation strategy is complex. This complexity emphasizes the necessity for ongoing discussions and developments within the ROS community to address these challenges and optimize callback scheduling within multi-threaded executors.

To assess the potential benefits and validate the practical utility of implementing partitioned and semi-partitioned scheduling in ROS 2 multi-threaded executors, we conducted an experiment based on a case study referenced in [8]. Our setup involved running a set of processing chains on a multi-threaded executor configured with four threads. For the fully partitioned scheduling scenario, we assigned three callbacks to each thread. For the semi-partitioned scenario, we specified thread affinities for each timer callback, while allowing the remaining callbacks to execute on any thread as per the default global scheduling policy. Figure 2a shows the processing chain configuration for our case study, where a lower chain index denotes a higher priority. We measured the response time of each chain as our primary metric to evaluate the effectiveness of both partitioned and semi-partitioned scheduling strategies within a multi-threaded executor. Preliminary results, depicted in Figure 2b, highlight the potential benefits of these alternative scheduling approaches.

IV. Conclusion

In this paper, we discussed the debate between different scheduling options within the context of ROS 2 and explored the advantages and drawbacks of each approach. Furthermore, we implemented partitioned and semi-partitioned scheduling for ROS 2 multi-threaded executors and conducted an experiment to encourage further investigation into the optimal scheduling options for ROS 2.

[1]Implementation details are available at https://github.com/rtenlab/ros2-picas.

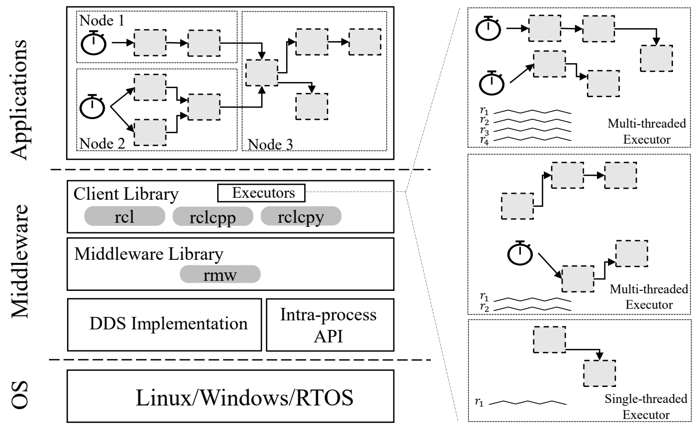
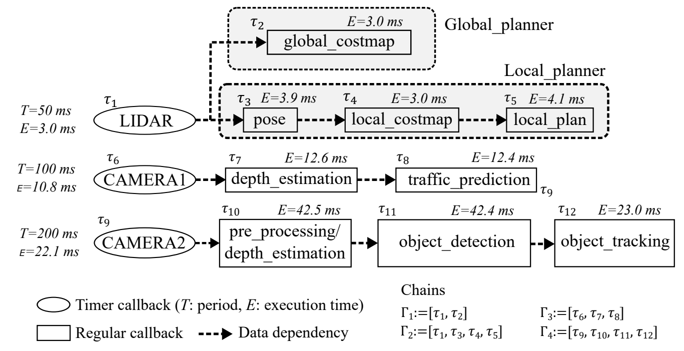
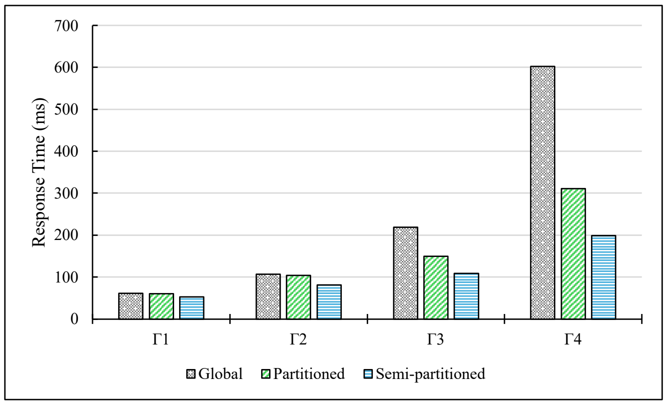


Fig. 1: ROS 2 architecture and application model [8]



(a) Chain set for case study [8]



(b) Comparison of response time

Fig. 2: An experiment of global, partitioned, and semi-partitioned scheduling in ROS 2

References

[1] D. Casini, T. Blaß, I. Lütkebohle, and B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *31st Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl, 2019, pp. 1–23.

[2] Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, and W. Yi, "Response time analysis and priority assignment of processing chains on ROS2 executors," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 231–243.

[3] H. Choi, Y. Xiang, and H. Kim, "PiCAS: New design of priority-driven chain-aware scheduling for ROS2," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 251–263.

[4] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, "A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 41–53.

[5] T. Blass, A. Hamann, R. Lange, D. Ziegenbein, and B. B. Brandenburg, "Automatic latency management for ROS 2: Benefits, challenges, and open problems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 264–277.

[6] A. A. Arafat, S. Vaidhun, K. M. Wilson, J. Sun, and Z. Guo, "Response time analysis for dynamic priority scheduling in ROS2," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 301–306.

[7] X. Jiang, D. Ji, N. Guan, R. Li, Y. Tang, and Y. Wang, "Real-time scheduling and analysis of processing chains on multi-threaded executor in ROS 2," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 27–39.

[8] H. Sobhani, H. Choi, and H. Kim, "Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2023, pp. 106–118.

[9] D. Enright, Y. Xiang, H. Choi, and H. Kim, "PAAM: A Framework for Coordinated and Priority-Driven Accelerator Management in ROS 2," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024.

[10] H. Choi, D. Enright, H. Sobhani, Y. Xiang, and H. Kim, "Priority-driven real-time scheduling in ROS 2: Potential and challenges," in *1st International Workshop on Real-time And intelliGent Edge computing (RAGE)*, 2022.

[11] H. Teper, T. Betz, M. Günzel, D. Ebner, G. von der Brüggen, J. Betz, and J.-J. Chen, "End-To-End Timing Analysis and Optimization of Multi-Executor ROS 2 Systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024.

[12] X. Luo, X. Jiang, N. Guan, H. Liang, S. Liu, and W. Yi, "Modeling and Analysis of Inter-Process Communication Delay in ROS 2," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 198–209.

[13] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 99–110.

[14] D. Casini, A. Biondi, and G. Buttazzo, "Task splitting and load balancing of dynamic real-time workloads for semi-partitioned edf," *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2168–2181, 2020.