Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors

Hoora Sobhani[*], Hyunjong Choi[†], Hyoseung Kim[*]

[*]University of California, Riverside

[†]San Diego State University

hsobh002@ucr.edu, hyunjong.choi@sdsu.edu, hyoseung@ucr.edu

29th IEEE Real-Time and Embedded Technology and Applications Symposium

May 9-12, 2023. San Antonio, Texas

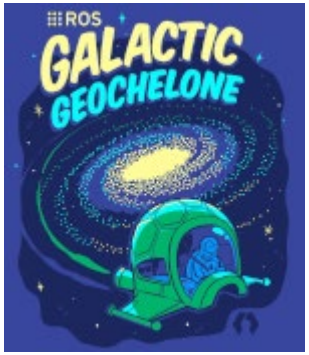
RTAS 2023

Introduction

What is ROS 2 ?

- ROS: open-source middleware framework
  - Software modularity and composability
  - Efficient and productive robotic software developments
  - ROS 2: the second generation of ROS

- Real-time capabilities in ROS 2
  - Significantly improved over its predecessor, ROS
  - But still incomplete for hard real-time and safety-critical systems
  - Needs support for strict timing guarantees

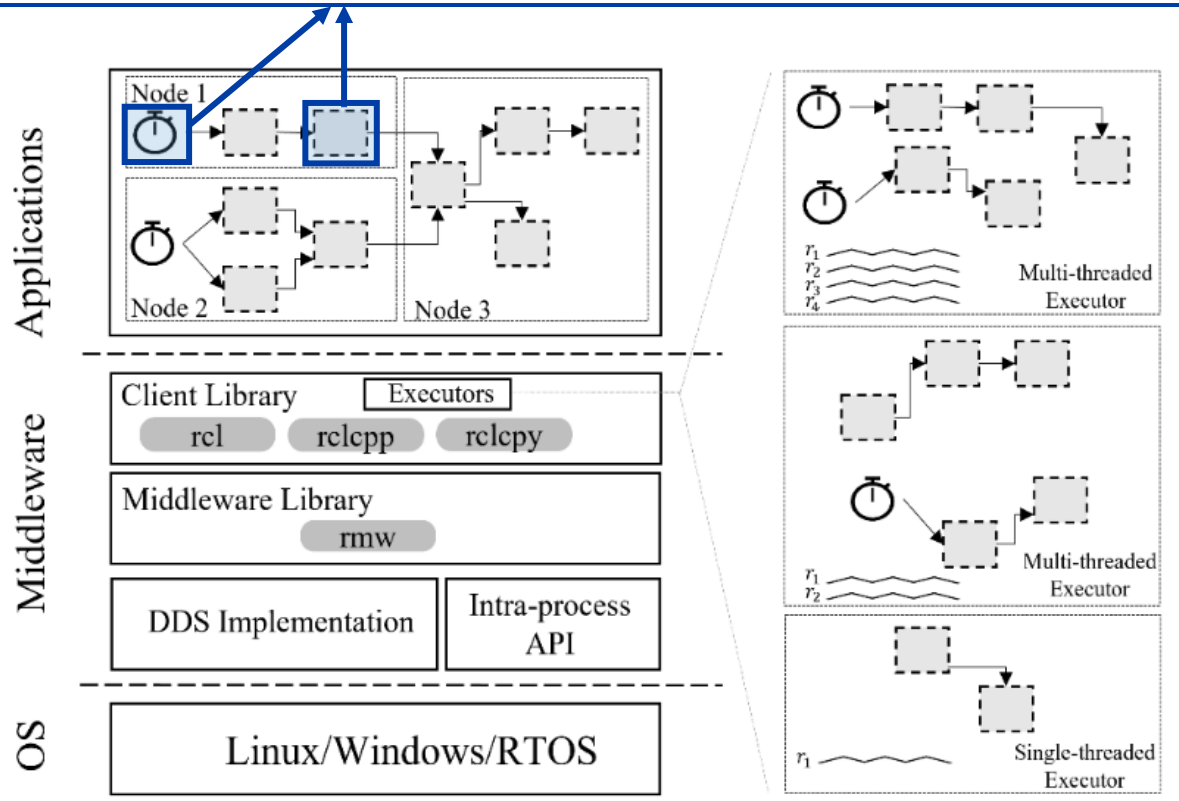


Introduction

ROS 2 Architecture

Callback: Minimal schedulable entity in ROS 2.

- Timer vs. Regular (subscription, service, client, waitable)
- Node: a collection of callback functions (for modularity and features partitioning)

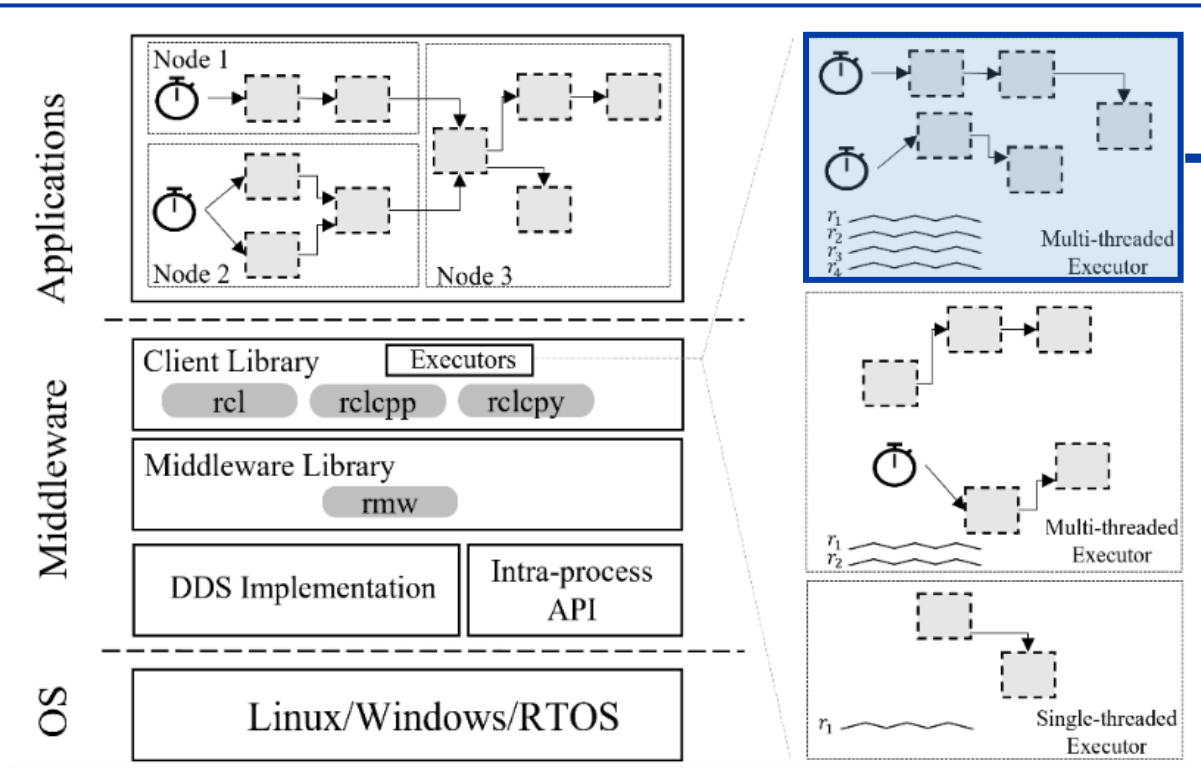


Introduction

ROS 2 Architecture

Executor: Abstraction of OS-level scheduling entities running on CPU cores

- **Single-threaded:** executes callbacks sequentially
- **Multi-threaded:** distributes pending callbacks across multiple threads (callbacks can execute in parallel)

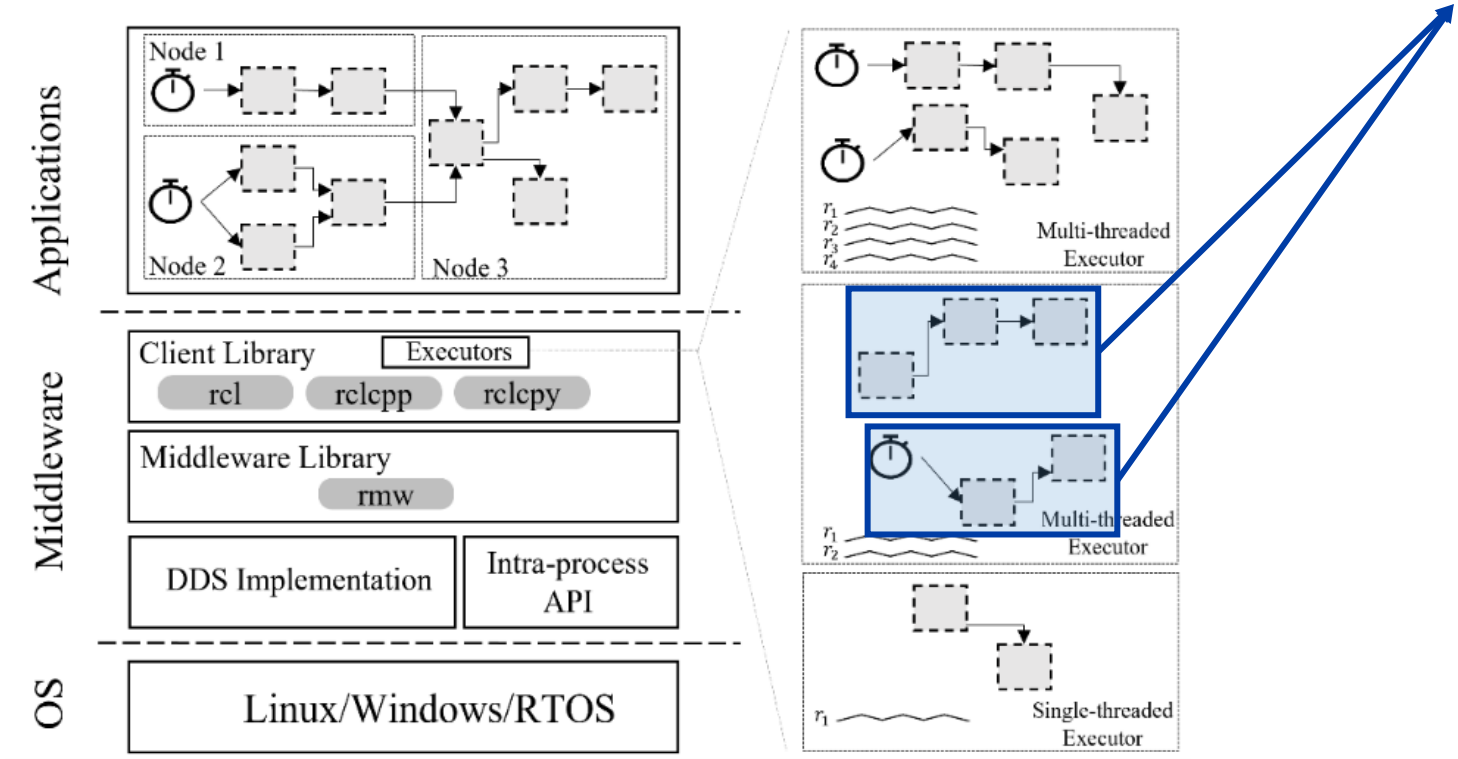


Introduction

ROS 2 Architecture

Callback Group: Controls the concurrency of callback execution.

- **Reentrant** callback groups allow an executor to execute any ready callback with no other restriction.
- **Mutually-exclusive** callback group option limits any callback within a group not to be executed in parallel.

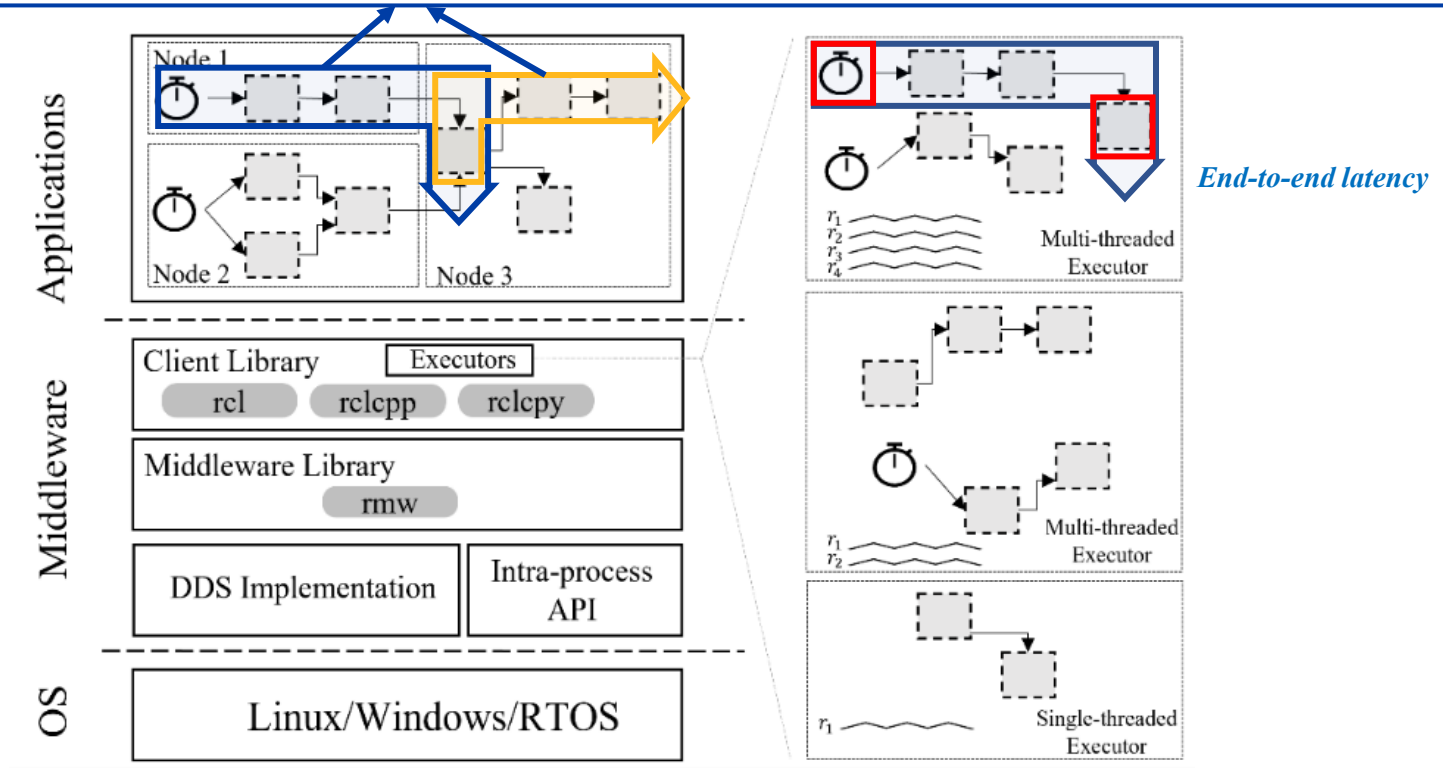


Introduction

Processing Chain

Chain: A sequence of callbacks with data dependencies.

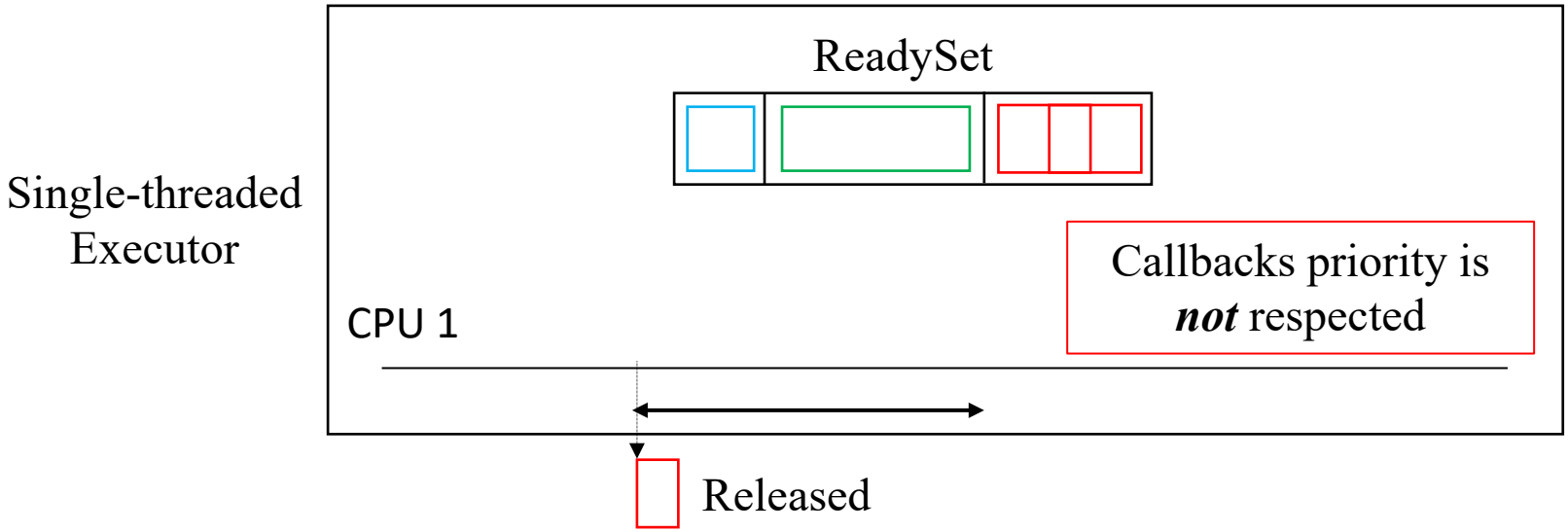
- **Chain priority:** criticality or importance of the chain in the system
- **End-to-end latency:** time interval between the release time of the first callback of a chain and the finish time of the last callback of the chain



Introduction

Callback scheduling in ROS 2 executors

- Callbacks are **non-preemptive**.
- Each executor maintains **one ReadySet:**
  - A cached set of "ready" regular (non-timer) callbacks.
  - Updated only when it is empty.
- An executor processes regular callbacks in its ReadySet plus incoming timer callbacks.



Introduction

ROS 2 multi-threaded executors

- Why do we need **multi-threaded** executor?
  - Improves system concurrency and throughput
  - Effectively utilizes multiple processors while preserving timing correctness
  - Example: real-time multi-threading in self-driving cars [Kim13]

Multi-threaded Executor

Related Work

- Real-time support for ROS
  - Real-time capabilities/performance improvement of ROS: [Saito18] [Gutierrez18] [Wei16] [Maruyama16]
  - Formal analysis on end-to-end latency of chains on ROS:[Choi20] [Abdullah19] [Kloda18] [Schlatow16]

- Formal timing analysis for ROS 2 (quite limited literature)
  - Single-threaded executor: [Arafat22] [Teper22] [Choi21] [Tang20] [Casini19]
  - Multi-threaded executor:
    - [Jiang22] Response-time analysis for ROS 2 multi-threaded executor for chains

Unique Contributions of Our Work

- Analyze end-to-end latency of processing chains on ROS 2 multi-threaded executors
  - Can be used for chains with both **constrained** and **arbitrary** deadlines
  - Supports callback groups (mutually-exclusive & reentrant)
- Improve chain latency with **priority-driven enhancements**
  - Enables priority-driven callback scheduling in executor
  - Presents priority-assignment
  - Provides analysis

Today's Outline

- [x] Introduction
- [x] Related Work
- [ ] Challenges
- [ ] Proposed Work
- [ ] Evaluation
- [ ] Conclusion

Challenges

- Why ROS 2 single-threaded analysis cannot be used?
- Why conventional non-preemptive scheduling not applicable?

Challenges

- **Why ROS 2 single-threaded analysis cannot be used?**
- Why conventional non-preemptive scheduling not applicable?

- **Polling Point (PP):** a time point when the ReadySet is empty (i.e., the executor is idle), the executor communicates with underlying layers to fetch new callbacks.
- **Processing Window (PW):** the time interval between two consecutive polling points.

[Tang20]:

Lemma 1: At most one instance of a regular callback executes in a PW.

Lemma 2: For a given chain, if the first callback of the k-th instance executes in PW_n, the earliest PW for the first callback of the l-th instance is PW_{n+l-k}.

Lemma 3: At most one instance of a regular callback from a chain instance executes in a PW.

Lemma 4: The instances of regular callbacks of a chain execute in consecutive PW.

PP and PW need to be redefined in multi-threaded executors

Some lemmas based on them do not hold any more!

Challenges

- Why ROS2 single-threaded analysis cannot be used?
- **Why conventional non-preemptive scheduling not applicable?**

ReadySet management:

- Unpredictable delay between release time of a callback and being ready to execute.
- Unknown taskset for each PW
- One callback instance only in the set
- Backlogged for multiple PW

Challenges

Can we extend single-threaded executor analysis?

- **Polling Point (PP):** the time when at least one thread becomes idle.
- **Processing Window (PW):** the time interval between two consecutive PPs, regardless of which thread triggered the new PP.
- **We revised the lemmas based on the new definitions of PP and PW** (Please see our paper for details).
- **But still many challenges remain!!**
  - When a new PP happens?
  - How long a PW takes?
  - How many PWs each callback would take to complete its execution?
  - How many PWs exist between two consecutive callbacks?
- **The difficulty multiplies for arbitrary-deadlines chains !!**

Our Approach:

Extending the conventional non-preemptive global task scheduling analysis while taking into account semantic differences introduced by chains, callback dependencies, and the ReadySet management

Proposed Work

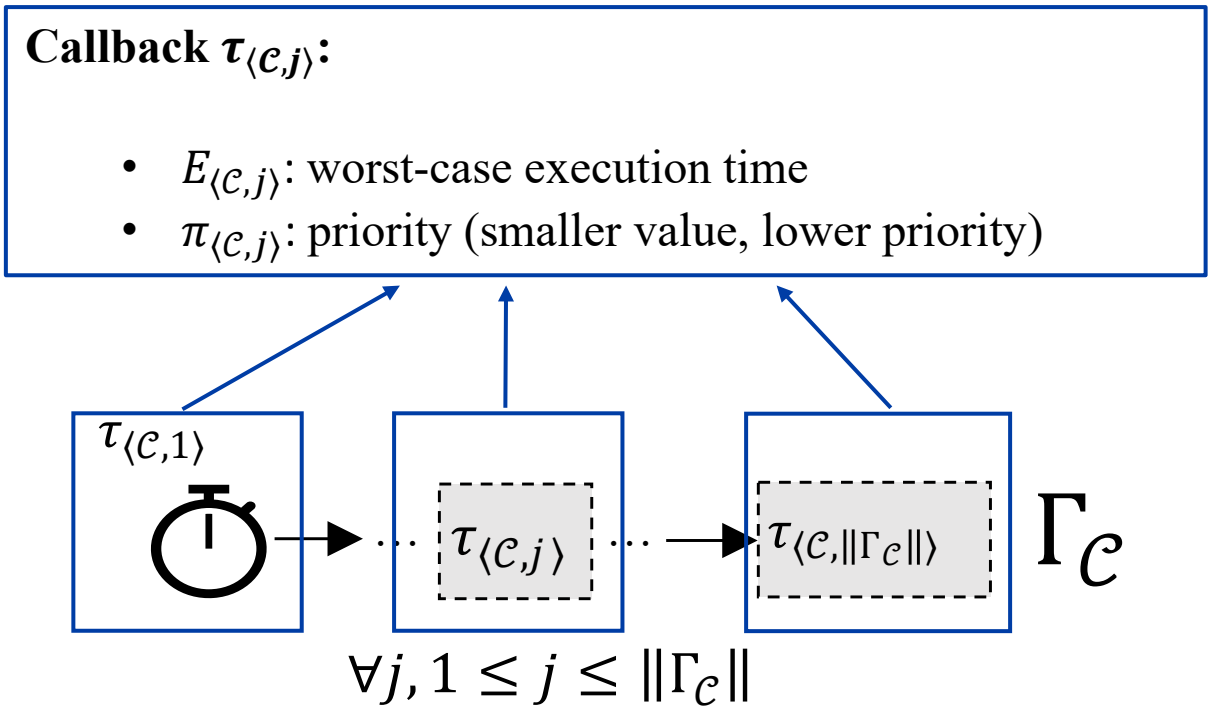
Response Time Analysis (RTA) framework for ROS 2 Multi-threaded Executor

	ROS 2 multi-threaded executor	
	Standard	Priority-driven
Constrained deadline	✓	✓
Arbitrary deadline	✓	✓
Reentrant callback group	✓	✓
Mutually-exclusive callback group	✓	✓
Multiple heterogenous executors	✓	✓

Today's Outline

- [x] Introduction
- [x] Related Work
- [x] Challenges
- [ ] Proposed Work
- [ ] Evaluation
- [ ] Conclusion

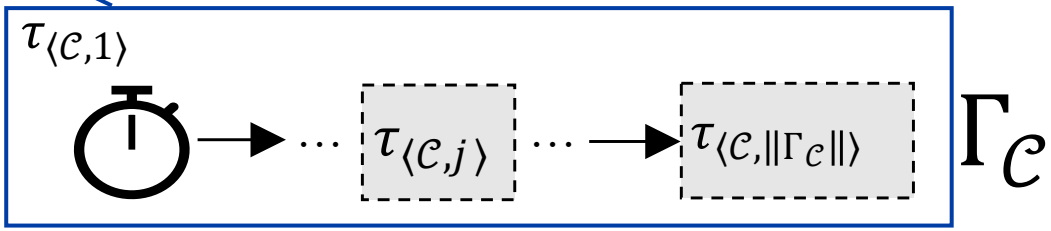
System Model



System Model

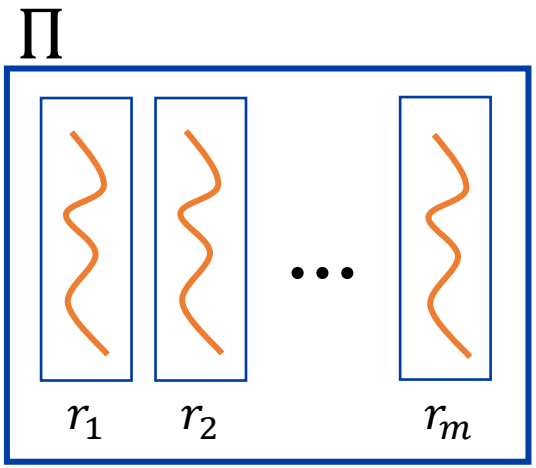
Chain $\Gamma_{\mathcal{C}}$: i-th instance: $\Gamma_{\mathcal{C}}^{i} = \{\tau_{\langle \mathcal{C},1 \rangle}^{i}, \tau_{\langle \mathcal{C},2 \rangle}^{i}, \tau_{\langle \mathcal{C},3 \rangle}^{i}, \ldots, \tau_{\langle \mathcal{C},\|\Gamma_{\mathcal{C}}\| \rangle}^{i}\}$

- $E_{\mathcal{C}}$: cumulative WCET of an instance of the chain
- $T_{\mathcal{C}}$: period of an instance
- $D_{\mathcal{C}}$: relative deadline of an instance
- $\pi_{\mathcal{C}}$: priority (smaller value, lower priority)



$$\forall j, 1 \leq j \leq \|\Gamma_{\mathcal{C}}\|$$

System Model



Multi-threaded Executor Π:

$$\Pi = \{r_1, r_2, \ldots, r_m\}$$

- ***m*:** number of threads ($1 \leq k \leq m$)
- Supply bound function of r_k [Shin08]

$$sbf_k(\Delta) = \begin{cases} \frac{C_k^r}{T_k^r}(\Delta - 2(T_k^r - C_k^r)) & \text{if } \Delta \geq 2(T_k^r - C_k^r) \\ 0 & \text{otherwise} \end{cases}$$

- Supply bound function of Π [Shin08]

$$sbf_\Pi(\Delta) = \sum_{r_k \in \Pi} sbf_k(\Delta)$$

Review of Conventional NP-FP Global Task Scheduling

- **Schedulability Test & Task Response Time:**

Needed resources Available resources

$$dbf(\Delta) < sbf_{\Pi}(\Delta)$$

Response Time

$$R_i = \Delta + \overline{sbf}_k(E_i - 1)$$

- **NP-FP scheduling [Lee14][Lee17]**

Workload of higher-priority tasks

$$dbf(\Delta) = \sum_{\tau_h \in hp(\tau_i)} W_h(\Delta, R_h - E_h) + \sum_{\tau_l \in mlp(\tau_i)} \min(E_l - 1, \Delta)$$

Workload of lower-priority tasks

- **Workload Function**

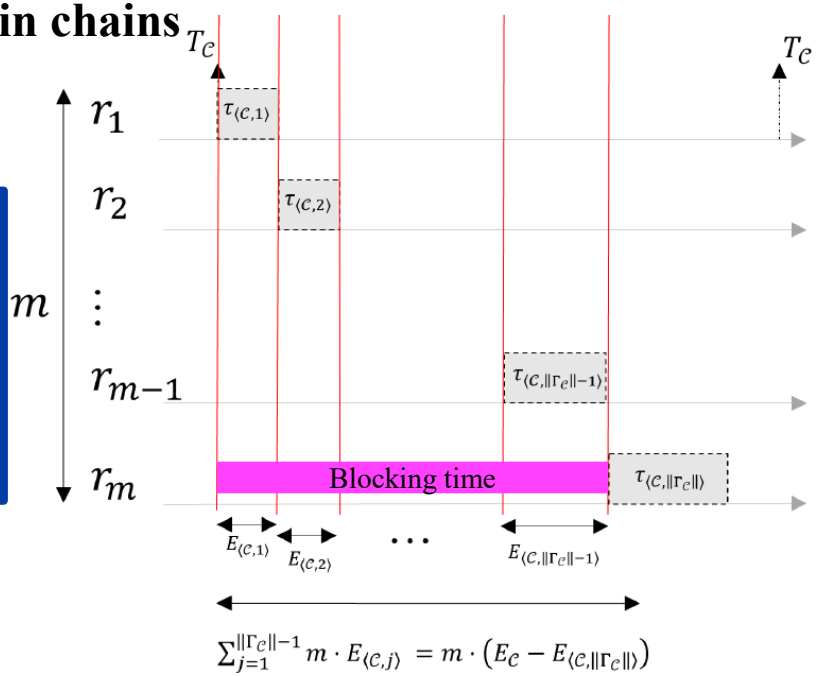
Workload of complete jobs Workload of the incomplete job

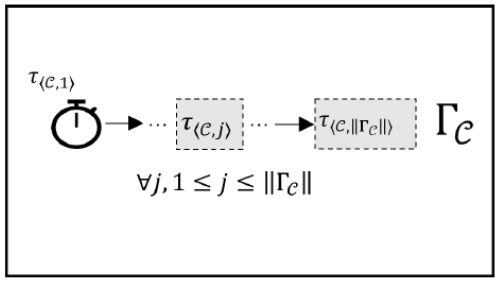
$$W_j(\Delta, \alpha) = \lfloor \frac{\Delta + \alpha}{T_j} \rfloor \cdot E_j + \min\left(E_j, \Delta - \lfloor \frac{\Delta + \alpha}{T_j} \rfloor \cdot T_j\right)$$

Key Considerations for ROS 2 Analysis

(1) Precedence dependency in chains

Next callback cannot start execution until its previous callback is completed, even if there exist other idle threads in the executor.





(2) Workload function

- Need separate workload functions for chains and callbacks
- Constrained vs. arbitrary deadlines: additional interference when AD is allowed
- Impact of callback groups

(3) ReadySet management

- Updated only at PPs → Multiple times of blocking by lower-priority callbacks

Response Time Analysis

	ROS 2 multi-threaded executor	
	Standard	Priority-driven
Constrained deadline	✓	✓
Arbitrary deadline	✓	✓
Reentrant callback group	✓	✓
Mutually-exclusive callback group	✓	✓
Multiple heterogenous executors	✓	✓

(see our paper for other cases)

RTA for a *standard* ROS 2 multi-threaded executor, *constrained-deadline* chains

- **Chain** Response Time:

Remaining execution of the last callback

$$R_C = \Delta + \overline{sbf}_k\left(E_{\langle c, \|\Gamma_C\|\rangle} - 1\right)$$

Executor's sbf

Due to precedence dependency

$$dbf(\Delta) = m \cdot \left(E_C - E_{\langle c, \|\Gamma_C\|\rangle}\right) + \sum_{\forall \Gamma_x \in \Gamma - \{\Gamma_C\}} W_x(\Delta, D_x - E_x)$$

Due to interfering chains

- Demand bound function of **Standard ROS 2 Multi-threaded Executor**

- **Chain** Workload Function for **CD**

Chain T & E

Due to an instance which carries out of Δ

$$W_{C'}(\Delta, \alpha) = \left\lfloor \frac{\Delta + \alpha}{T_{C'}} \right\rfloor \cdot E_{C'} + \min\left(E_{C'}, \Delta - \left\lfloor \frac{\Delta + \alpha}{T_{C'}} \right\rfloor \cdot T_{C'}\right)$$

Priority-driven Enhancement

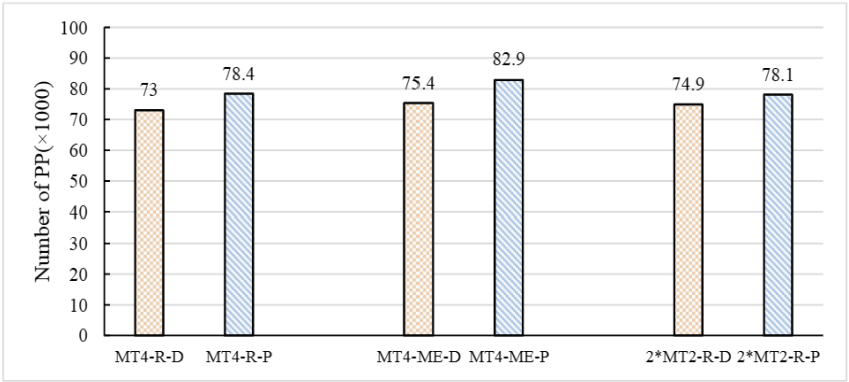
- **Motivation:** ReadySet management in a standard ROS 2 executor ignores the priorities.

$$dbf(\Delta) = m \cdot (E_C - E_{\langle C, \|\Gamma_C\| \rangle}) + \sum_{\forall \Gamma_x \in \Gamma - \{\Gamma_C\}} W_x(\Delta, D_x - E_x)$$

  - Lower-priority chains contribute as interference to higher-priority chains!

- **Our approach:** Make the executor strictly respect the priority of the corresponding chains when scheduling individual callbacks
  - Assigning callback priorities based on their respective chain priorities
  - Scheduling ready callbacks by strictly based on their priorities

- **Runtime Overhead:**
  - Updating ReadySet every time → Cancels the caching ability of ReadySet
  - Less than 10% of additional ReadySet updates compared to the default scheme (Our evaluation)



Overhead w.r.t. the number of polling points (PP)

Response Time Analysis

	ROS 2 multi-threaded executor	
	Standard	Priority-driven
Constrained deadline	✓	✓
Arbitrary deadline	✓	✓
Reentrant callback group	✓	✓
Mutually-exclusive callback group	✓	✓
Multiple heterogenous executors	✓	✓

(see our paper for other cases)

RTA for a *priority-driven* ROS 2 multi-threaded executor, *constrained-deadline* chains

- Demand bound function of **Priority-driven ROS 2 Multi-threaded Executor**

$$dbf(\Delta) = m \cdot (E_{\mathcal{C}} - E_{\langle \mathcal{C}, \|\Gamma_{\mathcal{C}}\| \rangle}) + \sum_{\forall \Gamma_x \in \Gamma - \{\Gamma_{\mathcal{C}}\} \wedge \pi_x > \pi_{\mathcal{C}}} W_x(\Delta, D_x - E_x) + \sum_{\forall \tau_l \in mlp(\tau_{\langle \mathcal{C},1 \rangle})} \min(E_l - 1, \Delta)$$

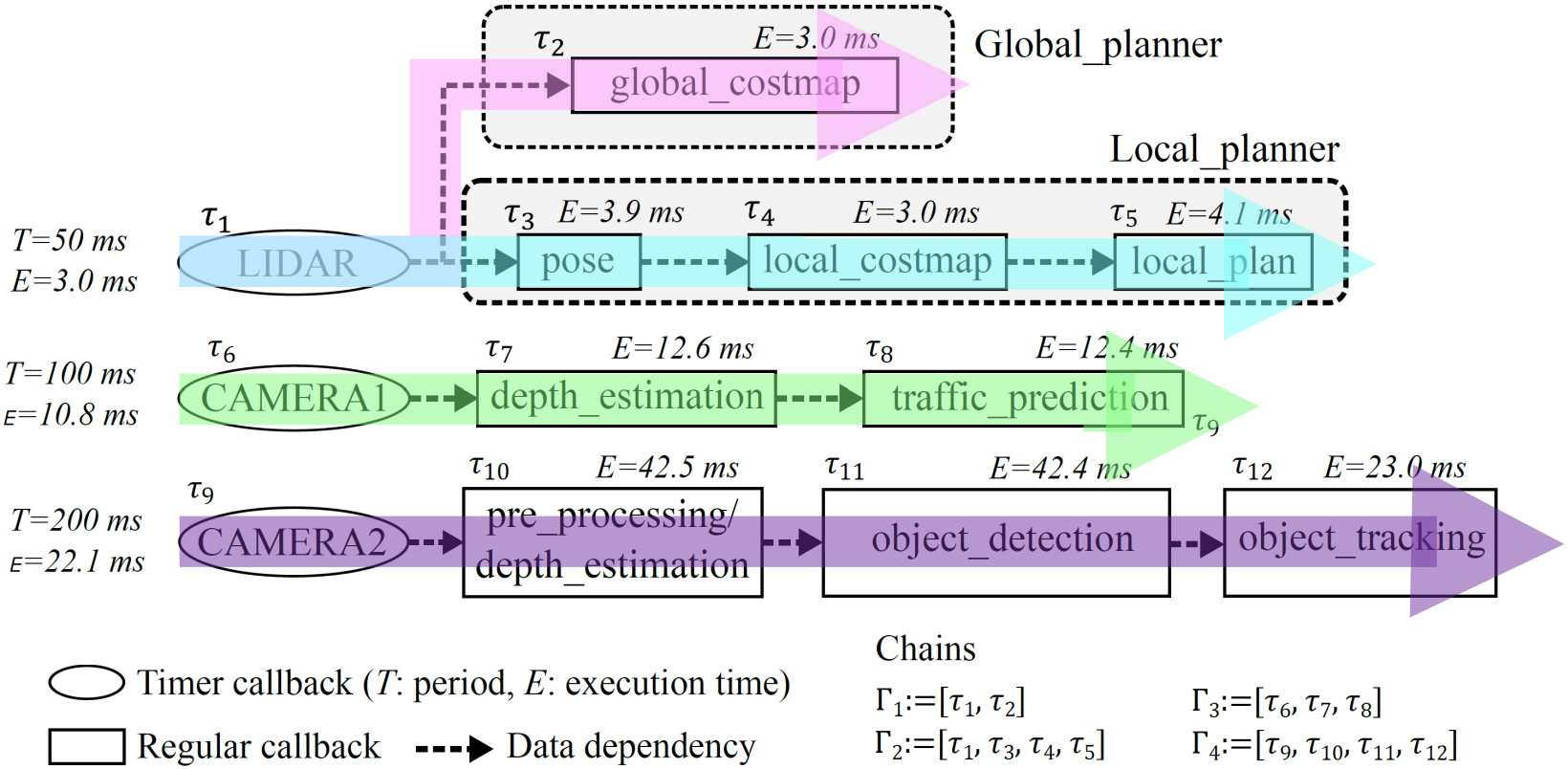
m largest lower-priority callbacks
(at most one callback from each chain)

Today's Outline

- ☑ Introduction
- ☑ Related Work
- ☑ Challenges
- ☑ Proposed Work
- ☐ Evaluation
  - ☐ Case study
  - ☐ Schedulability experiments
- ☐ Conclusion

Case Study

- ROS 2 (Galactic)
- NVIDIA Jetson AGX Xavier (AGX) platform (4 CPU cores , maximum frequency)
- Enhanced *rclcpp* package of ROS 2 with priority-driven scheduling

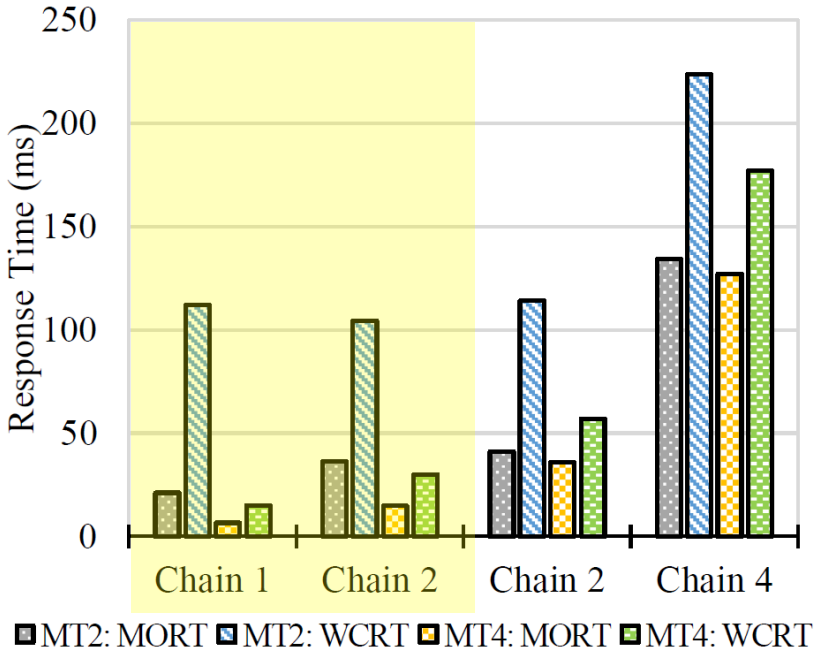


Case Study

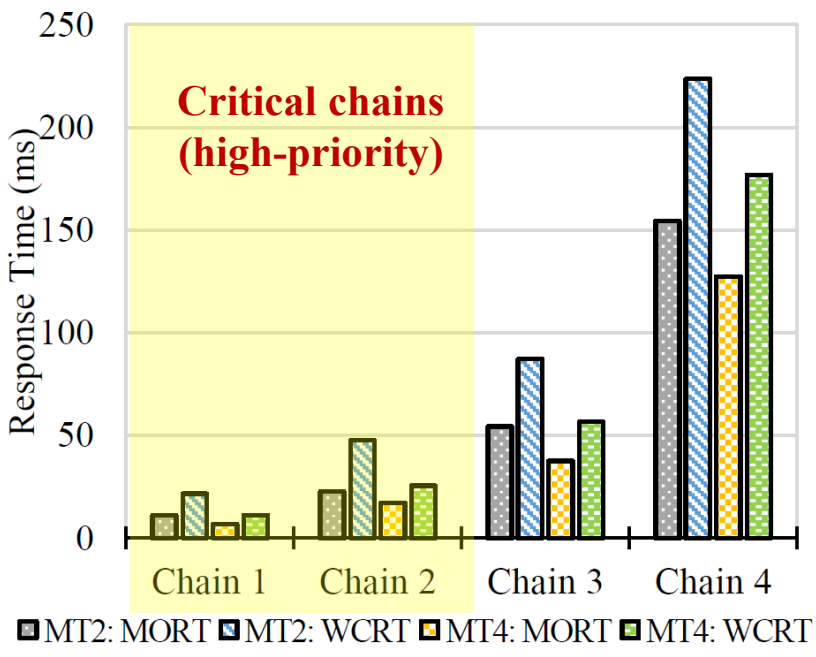
- **Comparison of observed and analyzed response times**

***Reentrant callback groups*
(see our paper for mutually-exclusive groups)

Standard ROS 2 Executor



Priority-Driven ROS 2 Executor



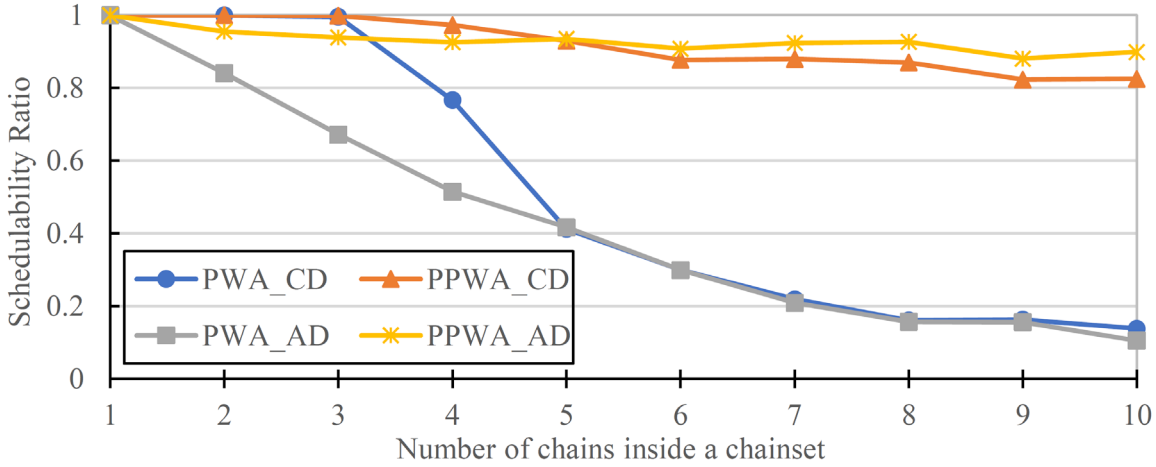
Observations:

- Our analysis safely upper-bounds the observed response times
- Priority-driven scheduling reduces analysis pessimism and improves RT of critical chains at runtime

Schedulability Experiments

- **Schedulability ratio by varying utilization , # threads, and # chains**

• **PWA_CD:** Proposed Worst-case Analysis (PWA) for chains with Constrained Deadlines (CD) – Theorem 1
• **PPWA_CD:** Priority-driven PWA for CD – Theorem 2
• **PWA_AD:** PWA for Arbitrary Deadlines (AD) – Theorem 3
• **PPWA_AD:** Priority-driven PWA for AD – Theorem 4



m = 1, UUnifast (D=T, D=2T),
U = 1, 1000 chainsets
Each chain has 10 callbacks

Observations:

- ROS 2 Priority-Driven vs. Default Scheduler:
  - 66% higher in schedulability
- Higher pessimism in ADs.
- Interference increases with # of chains
- Priority-driven scheduling is less effected
- Limited interference, only from HP chains

Today's Outline

- ☑ Introduction
- ☑ Related Work
- ☑ Challenges
- ☑ Proposed Method
- ☑ Evaluation
- ☐ Conclusion

Conclusion

- **Comprehensive RTA framework for ROS 2 multi-threaded executors**
- **Priority-driven scheduling as an improvement over the default ROS 2 policy**
  - Reduces response time of critical chains and improves analysis results
- **Case study on NVIDIA Jetson AGX Xavier and Schedulability experiments**

	ROS 2 multi-threaded executor	
	Standard	Priority-driven
Constrained deadline	✓	✓
Arbitrary deadline	✓	✓
Reentrant callback group	✓	✓
Mutually-exclusive callback group	✓	✓
Multiple heterogenous executors	✓	✓

Thank you

for your attention