PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS2

Hyunjong Choi, Yecheng Xiang, Hyoseung Kim
Robot Operating System (ROS)

- ROS (since 2007)
  - Popular open-source middleware in academia and industry
  - Provides software tools, robot systems, and best-practices

Over the decades, it has revealed shortcomings in real-time support for timing- and safety-critical applications
Why real-time in ROS?

- To develop safety-critical application with ROS
  - Autonomous driving software (e.g., autoware.ai)

---

Timing constraint violations (e.g., end-to-end latency) can cause catastrophic accidents

---

ROS 2 (since 2017)

- Most concepts are inherited from the original ROS design (e.g., pub-sub)
- Aims to improve **real-time capability, QoS, and security**
- Supports Data Distribution Service (DDS)

---

**Suffers from priority inversion**

**Too complex and pessimistic to analyze**

**No systematic resource allocation policy**

Needs a new RT scheduler for ROS2!
Contributions

- We propose a **new priority-driven chain-aware scheduler** for ROS2 in a multi-core environment (PiCAS)
- We develop analysis to **upper-bound the end-to-end latency of chains** under the proposed PiCAS framework
- We **implement PiCAS** in the Eloquent Elusor version of ROS2 on an **embedded platform** (NVIDIA Xavier NX)
- PiCAS outperforms the default ROS2 scheduler and the latest analysis work in terms of **end-to-end latency**
Scheduling-related abstractions in ROS2

- Callbacks, nodes, and executors

**Callback model**
- Timer / regular callbacks
- Non-preemptive
- $\tau_i := (C_i, D_i, T_i, \pi_i)$

**Chain model**
- $\Gamma^c := [\tau_s, \tau_{m1}, ..., \tau_e]$
- $\tau_s$ : the start callback of $\Gamma^c$
- $\tau_{m*}$ : the intermediate callback of $\Gamma^c$
- $\tau_e$ : the end callback of $\Gamma^c$

**Node model**
- $\mathcal{N} = \{n_1, ..., n_j, ..., n_N\}$
- Not schedulable entities

**Executor model**
- $\mathcal{E}_i = \{e_1, ..., e_j, ..., e_E\}$
- Preemptive
- Schedule with SCHED_FIFO

CPU cores

II. ROS 2 Background & System Model

II. ROS 2 Background & System Model
Challenges (1/2)

- Challenge I: Fairness-based scheduling within executors

<table>
<thead>
<tr>
<th>Single executor</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain 1</td>
<td>36.865</td>
<td>72.752</td>
<td>0.505</td>
<td>21.223</td>
</tr>
<tr>
<td>Chain 2</td>
<td>36.730</td>
<td>73.149</td>
<td>0.773</td>
<td>21.154</td>
</tr>
</tbody>
</table>

Semantic priority: Chain 1 > Chain 2

O1. Prioritizes timer callbacks regardless of chain priority†
O2. Does not distinguish callbacks by their origin chains

Jeopardizes the timeliness of safety-critical chains

† D. Casini et al. “Response-time analysis of ROS 2 processing chains under reservation-based scheduling”, ECRTS, 2019
Challenges (2/2)

- Challenge II: Priority assignment for executors

<table>
<thead>
<tr>
<th>Single executor</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain 1</td>
<td>0.370</td>
<td>0.392</td>
<td>0.366</td>
<td>0.004</td>
</tr>
<tr>
<td>Chain 2</td>
<td>48.795</td>
<td>97.783</td>
<td>0.772</td>
<td>28.304</td>
</tr>
</tbody>
</table>

< End-to-end latency results [sec] >

O3. High penalty due to self-interference
O4. No guidelines on executor priority assignment

Default ROS2 causes unacceptably high latency for chain 2
Priority-driven chain-aware scheduling

- Re-design ROS2 default scheduling architecture
  (1) Higher-semantic priority chain executes first (from challenge I)
  (2) For each chain, its instances on the same CPU execute in arrival order to prevent self-interference (from challenge II)

Lemma 1
For $\Gamma^c := [\tau_1, \ldots, \tau_i, \ldots, \tau_j, \ldots, \tau_N]$ whose callbacks are on the same CPU, a prior chain instance is guaranteed to complete, if the following conditions are met:
① $\tau_j$ has a higher callback priority than $\tau_i$,  
② $\tau_j$ runs on an executor with the same or higher priority than $\tau_i$’s executor.
Scheduling strategies

- Strategies for chains running within an executor

<table>
<thead>
<tr>
<th></th>
<th>Regular callbacks only</th>
<th>Timer and regular callbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single chain</strong></td>
<td><strong>Strategy I.</strong> (To satisfy ⊗ of Lemma 1)</td>
<td><strong>Strategy II.</strong> (To satisfy ⊗ of Lemma 1)</td>
</tr>
<tr>
<td></td>
<td>τ₁ → τ₂ → τ₃</td>
<td>τ₁ → τ₂ → τ₃ → τ₄</td>
</tr>
<tr>
<td><strong>Multiple chains</strong></td>
<td><strong>Strategy III.</strong></td>
<td><strong>Strategy IV.</strong></td>
</tr>
<tr>
<td></td>
<td>τ₁ → τ₂ → τ₃</td>
<td>τ₁ → τ₂ → τ₃ → τ₄</td>
</tr>
<tr>
<td></td>
<td>Chain 1</td>
<td>Chain 1</td>
</tr>
<tr>
<td></td>
<td>τ₁ → τ₂ → τ₃</td>
<td>τ₁ → τ₂ → τ₃ → τ₄</td>
</tr>
<tr>
<td></td>
<td>Chain 2</td>
<td>Chain 2</td>
</tr>
</tbody>
</table>

- Strategies for chains running across executors

<table>
<thead>
<tr>
<th>Single chain on one CPU</th>
<th>Multiple chains on one CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strategy V.</strong> (To satisfy ⊕ of Lemma 1)</td>
<td><strong>Strategy VI.</strong></td>
</tr>
<tr>
<td>τ₁ → τ₂</td>
<td>τ₃</td>
</tr>
</tbody>
</table>

10
Priority assignment

- **Realization of scheduling strategies** in two aspects
  - Callback priority assignment
  - Chain-aware node allocation algorithm

---

AlGORITHM 1 Callback priority assignment

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\Gamma \leftarrow$ sort in ascending order of semantic priority $\pi_{\Gamma}$</td>
</tr>
<tr>
<td>2</td>
<td>$p \leftarrow 1$ &gt; Initialize current priority</td>
</tr>
<tr>
<td>3</td>
<td><strong>for all</strong> $\Gamma^c \in \Gamma$</td>
</tr>
<tr>
<td>4</td>
<td><strong>for all</strong> $\tau_i \in \Gamma^c$</td>
</tr>
<tr>
<td>5</td>
<td>$\tau_i \leftarrow p$</td>
</tr>
<tr>
<td>6</td>
<td>$p \leftarrow p + 1$</td>
</tr>
<tr>
<td>7</td>
<td><strong>end for</strong></td>
</tr>
<tr>
<td>8</td>
<td><strong>end for</strong></td>
</tr>
</tbody>
</table>
Chain-aware node allocation

- **Purpose:** minimize interference between chains
  1. allocate given nodes to executors, and then
  2. maps executors to available CPU cores

**Part A.** Allocate sorted nodes $\mathbb{N}$ to $e_e$ and $e_e$ to a feasible CPU

**Part B.** Allocate sorted nodes $\mathbb{N}$ to feasible $e_m$ when $e_e$ does not exist

**Part C.** Handle all leftover nodes that were not allocated to executors by Part A & B

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{N}$</td>
<td>Nodes</td>
</tr>
<tr>
<td>$\mathbb{N}$</td>
<td>A node set consists of callbacks of a chain $\Gamma^c$ ($U_N \leq 1$)</td>
</tr>
<tr>
<td>$e_e$</td>
<td>Empty executor</td>
</tr>
<tr>
<td>$U_{P_k}$</td>
<td>Utilization of CPU core $P_k$</td>
</tr>
<tr>
<td>$n$</td>
<td>A node that has the lowest priority callback of $\Gamma^c$ in $\mathbb{N}$</td>
</tr>
<tr>
<td>$e_m$</td>
<td>Non-empty executors</td>
</tr>
<tr>
<td>$M$</td>
<td>The number of $e_m$</td>
</tr>
<tr>
<td>$P$</td>
<td>The number of $P_k$</td>
</tr>
</tbody>
</table>
Examples of chain-aware scheduling

- With the same workload at page 7.

### Single executor Mean Max Min STD

<table>
<thead>
<tr>
<th>Chain</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain 1</td>
<td>0.436</td>
<td>0.506</td>
<td>0.368</td>
<td>0.038</td>
</tr>
<tr>
<td>Chain 2</td>
<td>1.196</td>
<td>1.738</td>
<td>0.741</td>
<td>0.348</td>
</tr>
</tbody>
</table>

### Executor per chain Mean Max Min STD

<table>
<thead>
<tr>
<th>Chain</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain 1</td>
<td>0.369</td>
<td>0.394</td>
<td>0.366</td>
<td>0.004</td>
</tr>
<tr>
<td>Chain 2</td>
<td>1.255</td>
<td>1.731</td>
<td>0.737</td>
<td>0.352</td>
</tr>
</tbody>
</table>

< All callbacks in a single executor >

< One executor per chain >

Significantly improved end-to-end latency under PiCAS
Analysis of end-to-end latency

Latency analysis in a multi-core system

- Segment $\Phi_i$: a subset of a chain on one CPU core
- Multiple segments if a chain executes over multiple CPU cores

Step 1: Computing the WCRT of each segment of a chain

- WCRT of a segment $\Phi_i$, $R_{c,i}^n$

Step 2: Adding the WCRT of all segments of the chain

- End-to-end latency of a chain, $L_{\Gamma^c}$

End-to-end latency of a chain, $L_{\Gamma^c}$

$$L_{\Gamma^c} = \sum_{\Phi_i \in \Gamma^c} R_{c,i}^n + S(\Gamma^c)$$

Execution time of callbacks for the segment

$$R_{c,i}^{n+1} \leftarrow B_i + \sum_{j : \tau_j \in \Phi_i} C_j + \eta_i \left( R_{c,i}^n, \tau_k \right) \times C_k$$

Blocking time from lower priority callback

$$\forall j : \tau_j \in \Phi_i$$

Interference from higher semantic priority chains

$$\forall k : \tau_k \in e(\Phi_i) \vee \tau_k \in \text{HP}$$

Blocking delay by prior instance

$$\tau_k \in e$$
Evaluation

- Case studies, schedulability analysis, and analysis running time

- Experimental setup for case study
  - Implemented in the Eloquent Elusor of ROS2 on Ubuntu18.04 on NVIDIA Xavier NX
  - Comparison of approaches
    - ROS2: ROS2 default scheduler with no analysis
    - ROS2-SD: ROS2 default scheduler with resource reservation and WCRT analysis
    - ROS2-PiCAS: proposed scheduler with end-to-end latency analysis

- Case study in a multi-core system
  - Inspired by the indoor self-driving stack of F1/10 vehicle
  - 6 real-time chains (18 callbacks) and 6 best-efforts chains in a 4-core system
  - Low-indexed chains are more critical chains

† D. Casini et al. “Response-time analysis of ROS 2 processing chains under reservation-based scheduling”, ECRTS, 2019
Case study

PiCAS outperforms on most real-time chains (e.g., 14ms vs 88ms for chain 1)

Schedules chains while respecting their semantic priority

Our latency analysis provides tighter upper-bounds for real-time chains
Schedulability experiments

- Workload generation
  - 1,000 randomly-generated workload sets of callbacks
  - Utilization from \{2.5, 3.0, 3.5\} for 4-core environment
  - 45 callbacks that forms 9 chains (i.e., 5 callbacks per chain)
  - Chain’s period (deadline) is chosen in the range [50, 1000] msec

- Schedulability ratio decreases as the utilization increase
- ROS2-PiCAS outperforms ROS2-SD for all utilization setups.
- ROS2-PiCAS prioritizes chains based on their semantic priority
Conclusion & Future work

Conclusion

- Proposed a priority-driven chain-aware scheduling and its end-to-end latency analysis framework
- New design of ROS2 scheduling includes scheduling strategies, priority assignment of callbacks, and chain-aware node allocation
- ROS2-PiCAS outperforms the existing ROS2 scheduling w.r.t. the end-to-end latency under practical scenarios

Future work

- Deploy PiCAS to more complex scenario, e.g., autoware.auto (built on ROS2)
Thank you

PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS2

- Hyunjong Choi, Yecheng Xiang, Hyoseung Kim

Q & A