Priority-Driven Chain-Aware Scheduling with PiCAS

October 2021

Dr. Hyunjong Choi
Postdoc
University of California, Riverside
I. Motivation

II. PiCAS framework

III. PiCAS on reference system
I. Motivation
Motivation

- ROS 2 executor scheduling

Problems
- Suffers from priority inversion
- No systematic resource allocation methods
- Complex and pessimistic to analyze
- Difficult to prioritize critical chains

† D. Casini et al. "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", ECRTS, 2019
II. PiCAS Framework
PiCAS: Priority-driven Chain-Aware Scheduling framework for ROS2

- **Key idea**: enables *prioritization of mission-critical chains* across complex abstraction layers of ROS 2
  - To minimize end-to-end latency
  - To ensure predictability even when the system is overloaded

- **PiCAS**: Executor + Resource Allocation Algorithms + Timing Analysis
  - **PiCAS executor**: priority-driven callback scheduling
  - **Resource allocation algorithms**
    - Callback Priority Assignment
    - Chain-Aware Node-to-Executor Allocation
    - Executor Priority Assignment
  - Backed by *formal end-to-end latency analysis*
PiCAS Algorithms

- Strategies for chains running within an executor

<table>
<thead>
<tr>
<th>Single chain</th>
<th>Multiple chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular callbacks only</td>
<td>Timer and regular callbacks</td>
</tr>
<tr>
<td><strong>Strategy I.</strong> (To satisfy $\mathcal{C}$ of Lemma 1)</td>
<td><strong>Strategy II.</strong> (To satisfy $\mathcal{C}$ of Lemma 1)</td>
</tr>
<tr>
<td>$T_1 \rightarrow T_2 \rightarrow T_3$</td>
<td>$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$</td>
</tr>
<tr>
<td><strong>Strategy III.</strong></td>
<td>Chain 1</td>
</tr>
<tr>
<td>$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$</td>
<td>Chain 2</td>
</tr>
<tr>
<td><strong>Strategy IV.</strong></td>
<td>Chain 1</td>
</tr>
<tr>
<td>$T_1 \rightarrow T_2 \rightarrow T_3$</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 $\mathcal{C}$ of Lemma 1)</td>
<td><strong>Strategy VI.</strong></td>
</tr>
<tr>
<td>$T_1 \rightarrow T_2$ or $T_1 \rightarrow T_3$</td>
<td>$T_1 \rightarrow T_2$ or $T_3$</td>
</tr>
</tbody>
</table>

< Chain-aware scheduling strategies >

< Node-to-Executor allocation >

< Priority assignment >

< End-to-end timing analysis >

For details, please see our paper:
PiCAS Executor (1/2)

- Implemented as an extension to the rclcpp wait-set executor
- PiCAS executor API

```cpp
// Set RT priority and CPU affinity of executor instance
daemon void Executor::set_executor_priority_cpu(int priority, int cpu);

// Enable/Disable priority-based callback scheduling
void Executor::enable_callback_priority();
void Executor::disable_callback_priority();

// Set callback priority
void Executor::set_callback_priority(rclcpp::TimerBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::SubscriptionBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::ServiceBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::ClientBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::WaitableBase::SharedPtr ptr, int priority);

// Spin for PiCAS (RT executor priority & CPU affinity)
daemon void SingleThreadedExecutor::spin_rt();
```

**Default parameters**

```cpp
class Executor
{ ...
#ifdef PICAS
  bool callback_priority_enabled = false;
  int executor_priority = 0;
  int executor_cpu = 0;
#endif ...
}
```

client.hpp, service.hpp, timer.hpp, subscription_base.hpp, waitable.hpp
### Implementation details

#### Pro: waiting time for high-priority callback can be minimized

#### Con: overhead; not good for high throughput of short, same-priority callbacks

---

**get_next_executable** of executor.cpp (PiCAS)

```cpp
Bool Executor::get_next_executable
{
    bool success = false;
    if (!success) {
        wait_for_work(timeout);
    }
    success = get_next_ready_executable(any_executable);
    ...}
```

**get_next_ready_executable** of executor.cpp (PiCAS)

```cpp
Bool Executor::get_next_ready_executable
{
    ... 
    memory_strategy_->get_next_waitable(any_exe, weak_nodes);
    if (any_exe.cb && highest_priority < any_exe.waitable->callback_priority)
        highest_priority = any_executable.waitable->callback_priority;
        any_executable.timer = nullptr;
        any_executable.subscription = nullptr;
        any_executable.service = nullptr;
        any_executable.client = nullptr;
    }
    else any_executable.waitable = nullptr;
    ...}
```

---

Callbacks can be scheduled based on their priorities

- **Pro**: waiting time for high-priority callback can be minimized
- **Con**: overhead; not good for high throughput of short, same-priority callbacks
III. PiCAS on reference system
PiCAS on reference system

- Clone our forked repository
  
  ```
  git clone https://github.com/rtenlab/reference-system.git
  ```

- Build with PiCAS executor
  
  - Use PICAS CMake variable
    
    ```
    colcon build --cmake-args -DRUN_BENCHMARK=TRUE -DTEST_PLATFORM=TRUE -DPICAS=TRUE
    ```

- Configuration change for Linux RT priority
  
  - Modify `/etc/security/limits.conf`
    
    ```
    <userid> hard rtprio 99
    <userid> soft rtprio 99
    ```

- Notes
  
  - PiCAS is implemented as an extension to `rclcpp`, located in `reference-system/rclcpp`. This local `rclcpp` overrides the default ROS2 rclcpp.
  
  - If `-DPICAS=FALSE`, `reference-system/rclcpp` is exactly the same as the ROS2 Galactic version.
How to use PiCAS executor

Declaration of executor

Enable callback priority (PiCAS executor)

Set executor's real-time priority and assign CPU

Allocate nodes to executor

Run executor (\textit{spin\_rt()})

Executor (process) scheduled under SCHED_FIFO policy that has a \texttt{sched\_priority} value in the range 1 (low) to 99 (high).

\begin{verbatim}
std::thread
spinThread(&rclcpp::executors::SingleThreadedExecutor::spin_rt, &executor);
\end{verbatim}

autoware_default_singlethreaded_picas_single_executor.cpp
How to assign callback priority

- Callback priority assignment on reference system
  
  **Set unique priority to callbacks**

  ```
  namespace callback {
    namespace priority {
      struct Default {
        // The higher number, more critical callback
        static constexpr int FRONT_LIDAR_DRIVER_CALLBACK = 51;
        static constexpr int REAR_LIDAR_DRIVER_CALLBACK = 50;
        static constexpr int POINT_CLOUD_MAP_CALLBACK = 22;
        static constexpr int LANE_FOLLOW_CALLBACK = 30;
        static constexpr int VISUALIZER_CALLBACK = 27;
        static constexpr int POINTS_TRANSFORMER_CALLBACK = 52;
        static constexpr int POINTS_TRANSFORMER_FRONT_CALLBACK = 53;
        static constexpr int POINT_CLOUD_FUSION_CALLBACK = 55;
        static constexpr int POINT_CLOUD_FUSION_CALLBACK = 54;
        static constexpr int POINT_CLOUD_MAP_LOADER_CALLBACK = 24;
        static constexpr int VOXEL_GRID_DOWN_SAMPLER_CALLBACK = 23;
        static constexpr int RAYGROUND_FILTER_CALLBACK = 56;
        static constexpr int ND_TLOCALIZER_CALLBACK = 26;
        static constexpr int ND_TLOCALIZER_CALLBACK = 25;
        static constexpr int EUCLIDEAN_CLUSTER_SETTINGS_CALLBACK = 47;
        static constexpr int EUCLIDEAN_CLUSTER_OUTPUT_CALLBACK = 49;
        static constexpr int EUCLIDEAN_CLUSTER_DETetection_CALLBACK = 57;
      }
    }
  }
  ```

  [autoware_reference_system/system/priority/default.hpp]

- Or, use API, e.g.,
  ```
  executor.set_callback_priority(node->callback, priority)
  ```

  [reference_system/include/reference_system/nodes/rclcpp/nodes.hpp]

  [autoware_reference_system/include/autoware_reference_system/autoware_system_builder.hpp]

  [roscon.ros.org/world/2021]
Evaluation

- **Autoware model**

  - Based on the PiCAS priority assignment and node-to-executor allocation algorithms
  - Algorithm implementation: [https://github.com/rtenlab/ros2-picas](https://github.com/rtenlab/ros2-picas)

- **Single executor instance & multiple executor instances**
  - Based on the PiCAS priority assignment and node-to-executor allocation algorithms
  - Algorithm implementation: [https://github.com/rtenlab/ros2-picas](https://github.com/rtenlab/ros2-picas)
Evaluation

- Experiment environment
  - Raspberry Pi 4 with fixed CPU frequency of 1.5GHz
  - 4 CPU cores for multiple executors (PiCAS) and multithreaded executor (ROS2 default)
  - Run with `RUN_TIMES` option of 60 seconds
  - Evaluation criteria: Key Performance Indicators (KPIs) of reference system
Evaluation

- Latency summary

![Latency Summary Diagram](image)

Latency Summary Table 50s [FrontLidarDriver/RearLidarDriver (latest) → ObjectCollisionEstimator]

<table>
<thead>
<tr>
<th>#</th>
<th>exec</th>
<th>type</th>
<th>latency</th>
<th>mean</th>
<th>high</th>
<th>top</th>
<th>bottom</th>
<th>std_dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>autoware_default_multithreaded</td>
<td>rmw_cyclonedds_cpp</td>
<td>49.9478</td>
<td>68.1978</td>
<td>90.1094</td>
<td>76.76550699</td>
<td>59.61009</td>
<td>5.57771</td>
</tr>
<tr>
<td>1</td>
<td>autoware_default_singlethreaded</td>
<td>rmw_cyclonedds_cpp</td>
<td>300.14</td>
<td>458.074</td>
<td>513.353</td>
<td>478.1154</td>
<td>438.0326</td>
<td>29.0414</td>
</tr>
<tr>
<td>2</td>
<td>autoware_default_singlethreaded_picas_multi_executors</td>
<td>rmw_cyclonedds_cpp</td>
<td>42.6191</td>
<td>46.3615</td>
<td>72.4256</td>
<td>52.65746</td>
<td>39.65554</td>
<td>6.87596</td>
</tr>
<tr>
<td>3</td>
<td>autoware_default_singlethreaded_picas_single_executor</td>
<td>rmw_cyclonedds_cpp</td>
<td>53.0755</td>
<td>63.3119</td>
<td>97.5617</td>
<td>104.5243</td>
<td>82.11370000</td>
<td>11.2053</td>
</tr>
<tr>
<td>4</td>
<td>autoware_default_staticsinglethreaded</td>
<td>rmw_cyclonedds_cpp</td>
<td>155.9668</td>
<td>347.0277</td>
<td>553.763</td>
<td>410.0831</td>
<td>283.9709</td>
<td>63.3581</td>
</tr>
</tbody>
</table>
## Evaluation

- **Dropped messages summary**

![Diagram showing comparison of message drop rates across different executor types: Default-singletreaded, PiCAS-singletreaded, Default-multithreaded, PiCAS-multithreaded, and Default-static singlethreaded.]

### Dropped Messages Summary Table

<table>
<thead>
<tr>
<th>#</th>
<th>owo</th>
<th>rmw_cyclonedds_cpp</th>
<th>type</th>
<th>low</th>
<th>mean</th>
<th>high</th>
<th>top</th>
<th>bottom</th>
<th>std_dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>autoware_default_multithreaded</td>
<td>dropped</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>autoware_default_singletreaded</td>
<td>dropped</td>
<td>0</td>
<td>0.585132</td>
<td>1</td>
<td>1.202931</td>
<td>0.53333</td>
<td>0.334789</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>autoware_default_singletreaded_picas_multi_executors</td>
<td>dropped</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>autoware_default_singletreaded_picas_single_executor</td>
<td>dropped</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>autoware_default_static_singletreaded</td>
<td>dropped</td>
<td>0</td>
<td>0.0282776</td>
<td>1</td>
<td>0.19340256908</td>
<td>0</td>
<td>0.165125</td>
<td>0</td>
</tr>
</tbody>
</table>
Evaluation

- Behavior planner jitter

![Behavior Planner Jitter Summary Table 60s](image)

<table>
<thead>
<tr>
<th>#</th>
<th>exe</th>
<th>type</th>
<th>period</th>
<th>low</th>
<th>mean</th>
<th>high</th>
<th>top</th>
<th>bottom</th>
<th>std_dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>autoware_default_multithreaded</td>
<td>rmw_cyclonedds_cpp</td>
<td>90.7153</td>
<td>99.9908</td>
<td>108.075</td>
<td>102.53037</td>
<td>87.93143</td>
<td>2.83947</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>autoware_default_singletthreaded</td>
<td>rmw_cyclonedds_cpp</td>
<td>89.0889</td>
<td>110.359</td>
<td>187.791</td>
<td>137.493</td>
<td>83.306899999</td>
<td>27.05033</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>autoware_default_singletthreaded_picas_multi_executors</td>
<td>rmw_cyclonedds_cpp</td>
<td>99.6448</td>
<td>99.9592</td>
<td>100.403</td>
<td>100.09359087</td>
<td>99.9947913</td>
<td>0.0944087</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>autoware_default_singletthreaded_picas_single_executor</td>
<td>rmw_cyclonedds_cpp</td>
<td>88.1241</td>
<td>100.002</td>
<td>111.957</td>
<td>103.45313</td>
<td>96.550869999</td>
<td>3.45113</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>autoware_default_static_singletthreaded</td>
<td>rmw_cyclonedds_cpp</td>
<td>84.1847</td>
<td>163.516</td>
<td>256.623</td>
<td>102.5484</td>
<td>135.2836</td>
<td>26.8324</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

- CPU usage summary

- Memory usage summary
Thank you

Q & A

https://github.com/rttenlab/reference-system
Challenges (1/2)

- **Challenge I: Fairness-oriented callback scheduling within executors**

  Chain criticality: Chain 1 > Chain 2

<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>

  O1. Timer callbacks always get the highest priority
  O2. No way to respect chain criticality

  Fairness-oriented scheduling can jeopardize the timeliness of mission-critical chains
Challenges (2/2)

- Challenge II: Priority assignment of 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>

Chain criticality: Chain 1 > Chain 2

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

Default Linux scheduler or naïve priority assignment can cause unacceptably high latency

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