Chain-Aware ROS Evaluation Tool (CARET)

Atsushi Hasegawa (Research Institute of Systems Planning, Inc.)
Keita Miura (EMB IV, Inc.)

This project is being conducted in cooperation with the University of Tokyo and TIER IV.
Background | ROS 2 Features

- ROS 2 provides:
  - Pub/Sub communication
  - Callback scheduler
  
  ⇒ ROS 2 realizes loosely-coupled, highly interoperable systems

- Appropriate designing allow for a variety of systems.

(a) Many hosts

(b) Single process

Example of implementation for parallel processing of nodes
Response time is important for safety-critical systems, but difficult to measure. Cooperation of many nodes is important for system to behave as desired.
Use cases for Evaluation

- Division of labor for performance evaluation and analysis.
  - Measure performance with simulation or field testing.
  - Analyze from multiple perspectives with various teams.
Introduction to CARET

- Our objective:
  - Evaluating response time
  - Involving various teams in tackling performance issues even for a large-scale ROS 2 system like Autoware.

- To achieve these objectives, we developed Chain-Aware ROS Evaluation Tool (CARET).

  For example, CARET answers:
  - “Autoware response time was X ms.”
  - “QoS history of /cmd_vel topic was X.”

https://github.com/tier4/CARET
Apache License 2.0
Features

- Lightweight
  - Add extra LTTng trace points via hook
  - Record only necessary information

- Flexibility
  - Provide Python-API for developers to analyze issues

- Intuitively evaluable visualization
  - Support several types of intuitive figures
Measurement Targets

- CARET measures:
  - Callback latency
  - Communication latency
  - Node latency
  - Path latency
Measurement Targets

- CARET measures:
  - Callback latency
  - Communication latency
  - Node latency
  - Path latency
Measurement Targets

- CARET measures:
  - Callback latency
  - Communication latency
  - Node latency
  - Path latency
Visualization Concept

CARET supports coarse to fine granularity visualization APIs.

Any issues?
- response time
  - min: 100 ms
  - ave: 150 ms
  - max: 300 ms

Is QoS history depth appropriate?

Were callbacks executed instantly?

- Callback scheduling
  - Callback
  - Callback

Message flow

Response time

Granularity

Coarse

Fine
1. Developers define node path to evaluate.
2. CARET draws message dependencies between I/O nodes.

However, latency in message flow is not suitable for response time evaluation.
Key Idea | Response time

"When does the system respond to events?"

1. CARET uses each first flow that reflects input message.
2. CARET considers worst case as well.
Diagrams and Figures

Diagram

- Visualize Message Flow
- Visualize Response Time in Histogram
- 1s (best)
- 2s (worst)

Figures by CARET

- Message flow of target
- Time [s]
- Probability
- Response Time [ms]
Evaluation Steps with CARET

Evaluation steps:
1. Recording
2. Configuration
3. Visualization
Evaluation Steps | Recording

- Generate trace data
  - Trace data is historical data and consists of information at trace points. (timestamp, trace point’s type, message address, etc)

### Recording

- `ros2 caret record` (command to record information)
- `ros2 launch package_name` (launch target applications)

<table>
<thead>
<tr>
<th>timestamp</th>
<th>type</th>
<th>args</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>callback_start</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>publish</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>callback_end</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Evaluation Steps | Configuration

- Create an “Architecture file” which defines followings
  - Target paths
  - Target application structure

Architecture File
- Target path
  - [node_name, topic_name, node_name, …]
- Executor Information
  - Type
- Node Information
  - Callbacks Information
  - Node Latency Definition
Evaluation Steps | Visualization

- Visualize the measurement results
  - CARET provides visualization APIs.
  - Developers can check the results with Jupyter-notebook.

```python
#!/usr/bin/env python3
from caret_analyze.plot import Plot
... # Processing trace data and architecture
Plot.create_path_histogram(target_path)

Plot.create_message_flow(target_path)

Plot.create_callback_sched(target_path)
```
Measurement of an Actual System

Autoware
- An open source ROS based autonomous driving system

Target path:
- From Sensing to Control
- Including major modules / nodes for autonomous driving
Measurement Results of Autoware

Target path Histogram

Each module Histogram

Node Histogram

Sensing → Localization → Planning → Control

Long Tail.
Future Work

Provide feedback on the perspectives of the autoware evaluation.

1. **Expand measurement coverage**
   - Path containing /tf topic
   - DDS layer
   - System call
   - Multiple host

2. **Mitigate limits and constraints**
   - Support complex nodes

3. **Propose to integrate CARET trace points into each official package**
Discussion and comments are welcome!

This presentation is based on results obtained from a project, JPNP16007, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).


https://github.com/tier4/CARET
Examples of Complex Nodes

Possible to measure

Sub
node latency definition

Pub

receive msg

publish msg
time

The node receiving data and publishing the data.

sub_cb = [](&msg){
  msg_ = func(msg);
  publish(msg_)
}

Impossible to measure

Sub

Pub

receive msg

publish msg
time

The node receiving some data and publishing one data.

sub_cb = [](&msg){
  queue.push(msg)
  msg_ = avg(queue);
  publish(msg_)
}

The node using the data at a specific time.

sub_cb = [](&msg){
  msg_ = lookup(msg.stamp);
  publish(msg_)
}
Overhead

Measurement Method

```cpp
msg_ ->data = "Hello World";
start_time = now();
pub_ ->publish(msg_);
end_time = now();

exe_time = end_time - start_time;
```

![Graph showing overhead of Publish function with box plot]
Output Figures with CARET

<table>
<thead>
<tr>
<th>System Level</th>
<th>Coarse</th>
<th>Visualization Granularity</th>
<th>Fine</th>
</tr>
</thead>
</table>

**Target Granularity**

**Callback Level**

- `/sensor_dummy_node`
  - topic1: min 0.02 ms, avg 2.06 ms, max 118.63 ms
- `/timer_node`
  - topic2: min 0.04 ms, avg 2.98 ms, max 26.54 ms
- `/message_driver_node`
  - topic3: min 0.02 ms, avg 0.36 ms, max 13.65 ms
- `/timer_driver_node`
  - topic4: min 0.03 ms, avg 1.14 ms, max 2.27 ms
- `/actuator_dummy_node`

Scalable
Trace Points

- Add trace points to record information.
  - Time (e.g. callback start)
  - Implementation (e.g. node name)
  - Configuration (e.g. QoS)
  - Message identifier (address, message stamp)

※ Skip recording by hook if nodes or topics are unnecessary.

```
$ ./main
int main {
  puts("hello");
}
```

```
$ LD_PRELOAD=libhook.so ./main
int main {
  puts("hello");
  // any code
  libc_puts(s);
}
```
Definition of target paths

- Define a path to evaluate from a node graph.

![Node graph with paths]

- Search to select a path
  - `search(A, E)` # CARET API
  - Enumerate the paths to be written in one stroke from node A to node E.
  - returns:
    - [A,C,E]
    - [A,D,E]

- Define manually
  - It is possible to specify the path manually.
Scheduling Visualization

- Callback scheduling affects node latencies.
- Scheduling visualization decompose node latencies into callback latencies and scheduling latencies.