ROS 2 アプリケーションのリアルタイム性検証の取り組み

CARET によるEnd-to-End レイテンシ評価

ROSCon JP 2021
2021/9/16

システム計画研究所
長谷川 敦史

本プロジェクトは東大様および Tier IV 様と合同で進めているプロジェクトです。

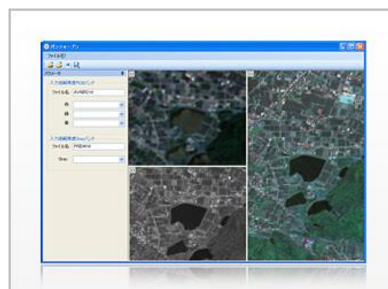
会社紹介

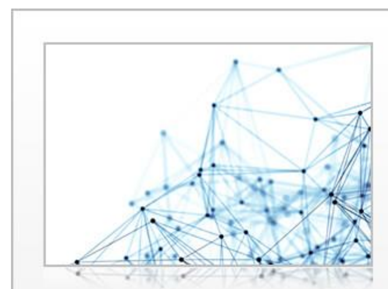
- （株）システム計画研究所（ISP）
  - 1977 年創業
  - 研究開発型のソフトウェア会社
- 事業分野

人工知能

医療情報

画像処理


宇宙・制御


通信
ネットワーク

[技術情報サイト]

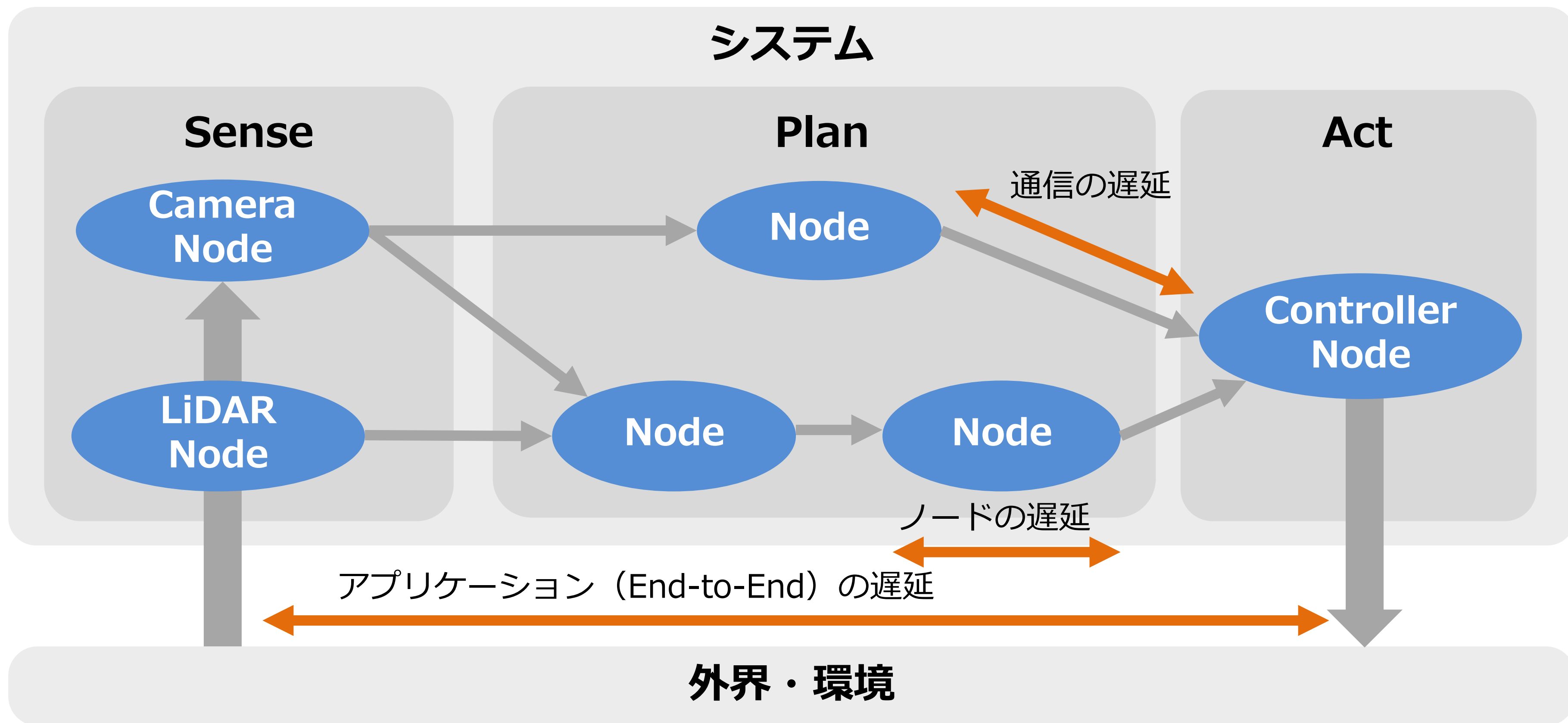
- 東大様および Tier IV 様と合同で Autoware の性能解析に取り組む

背景｜ROS に求められる性能要求

ISP

■ ロボット（ROS）は正常な動作のために、
パフォーマンスやリアルタイム性が求められる

所定の処理に時間要求があること



指針を立てるためにも、まずは測定することが重要

性能測定の手法｜トレース

- プログラムに測定用のソースコード埋め込み、コードが実行される度に記録をしていく方法

```
tracepoint(“start”);
task();
tracepoint (“end”);
```

実行時間 = $t_{end} - t_{start}$ [s]

- トレースポイントには付加情報をつけることもある

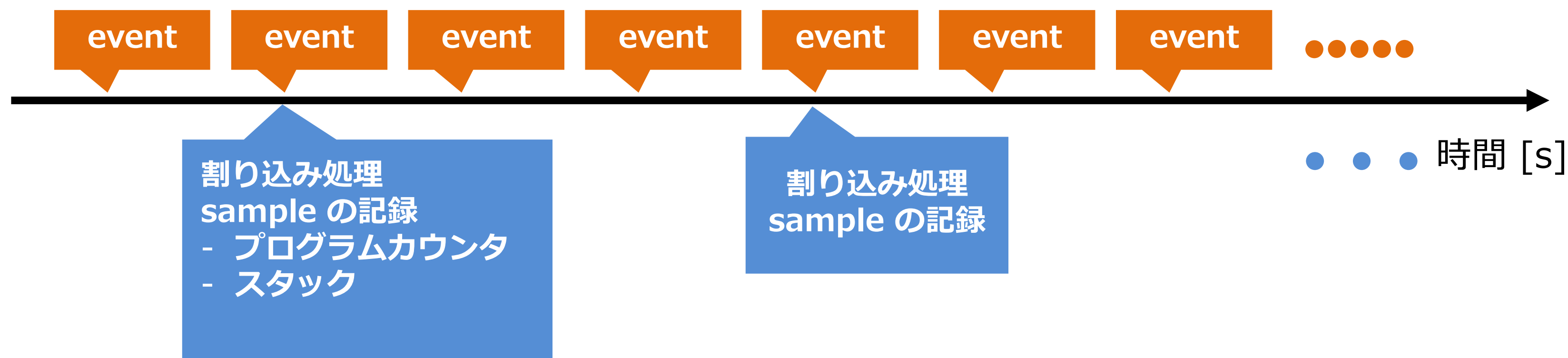
```
tracepoint(“publish”, データサイズ);
```

バンド幅 [Byte/s] = 1 秒間あたりのデータ量

性能測定の手法｜プロファイリング

ISP

■ 特定のイベントが一定数発生した際に実行状態を記録



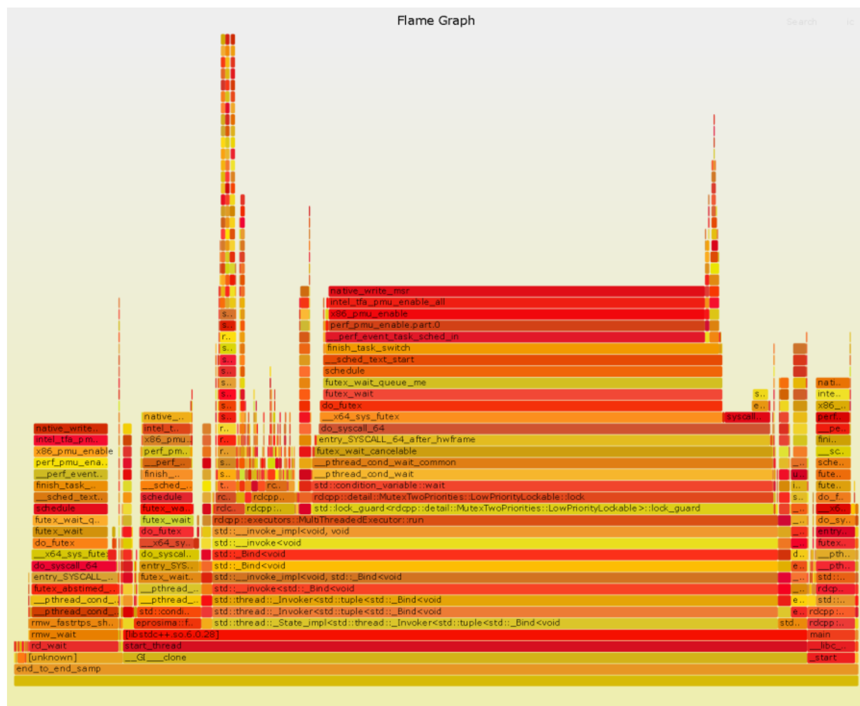
■ クロックを event とした場合、
実行時間の多い処理ほどサンプルが多くなる
⇒ ボトルネックがサンプル数の多い処理として検出される

■ event は キャッシュミス、ページフォルトなど指定可

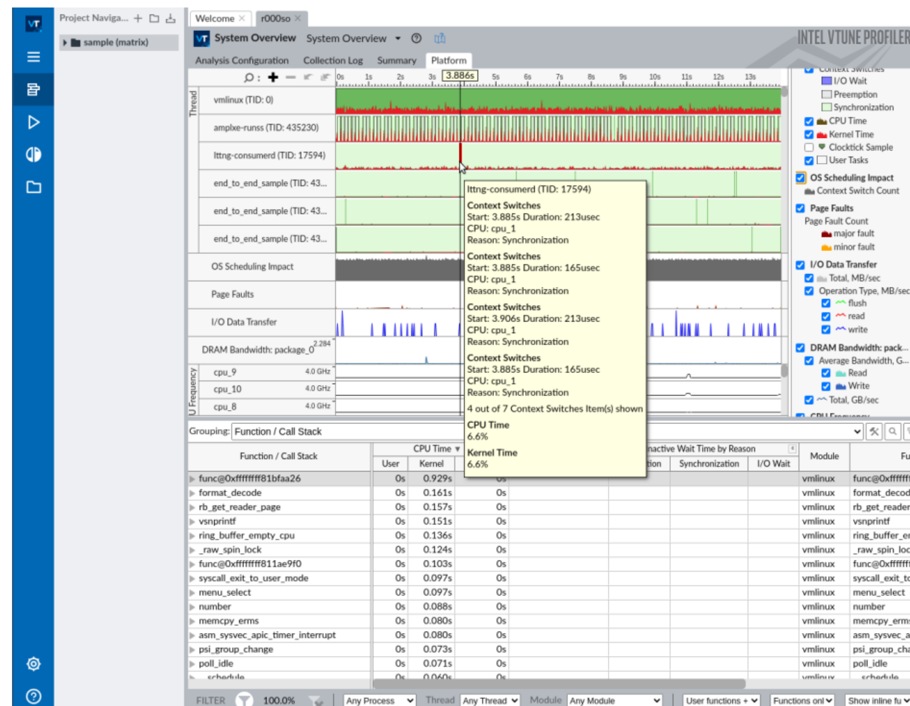
既存ツールの紹介｜Linux の測定ツール

ISP

- プロファイリングツール：perf, vtune, ...
- トレースツール：Linux Trace Toolkit ng, ...



perf の 出力例 （ Flame Graph ）



vtune の出力例 （System Overview）

- ボトルネックとなっている処理やリソース特定などに有用
- プロセス単位の測定なので、ノードの評価が困難なことも

既存ツールの紹介｜ROS 2 の測定ツール

ISP

■ ros2 topic

$ ros2 topic hz /topic　：メッセージ受信周波数 [Hz]

$ ros2 topic bw /topic　：メッセージのバンド幅 [Byte/s]

$ ros2 topic delay /topic　： 現在時刻 – msg.header.stamp [s]

※ メッセージの型に header が必要

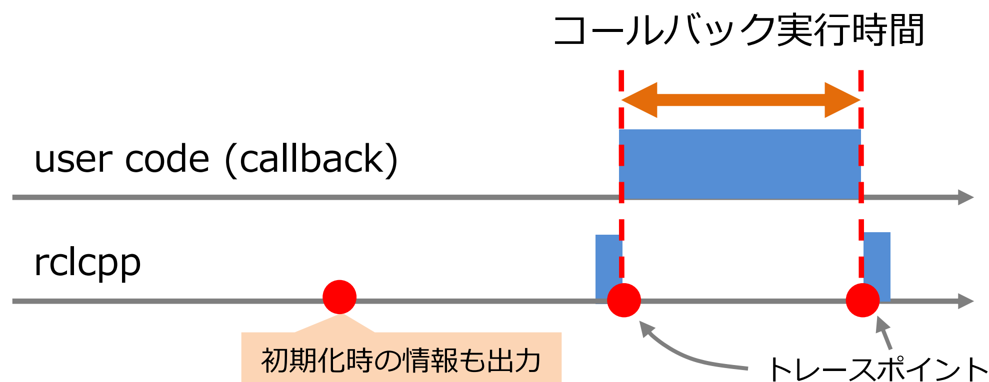
※ publish 時、現在時刻を msg.header.stamp に代入する必要あり

■ ros2 trace

$ ros2 trace　：トレースポイントの記録

$ ros2 trace-analysis　：記録したトレース結果の解析

※ Linux のみ。 ros2 trace 関連の追加インストールが必要



ros2 trace の主要なトレースポイントと測定対象

既存ツールの課題・新規ツール開発の動機

ISP

- 課題
  - Linux ツール：ROS アプリケーションの内部動作が把握しにくい
  - ROS ツール：評価対象が限定的・メッセージ型の制約
- 動機
  - ノードやアプリケーションの評価がしたい

    ⇒ **Chain-Aware ROS Evaluation Tool (CARET)** を開発

ros2 trace の拡張
Apache 2.0
※Galactic のみ対応

- 通信
- コールバック
- ノード
- アプリケーション

ROS ツール

CARET

測定対象

ROS アプリケーションのより詳細な評価が可能

[.msg]
Header header
***

ROS ツール

要 Header

[.msg]
***

CARET

任意の型

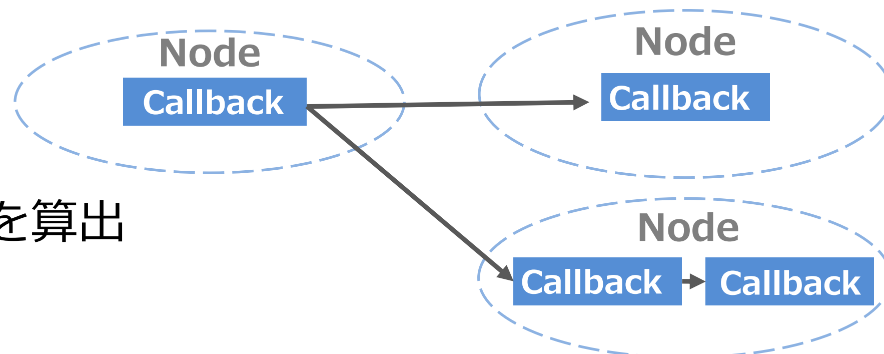
測定可能なメッセージの型

任意のメッセージ型の測定が可能

CARET | 測定対象

ISP

- メッセージの流れをトラッキング
- コールバックグラフを再構築し、
  トレース結果からノードレイテンシなどを算出



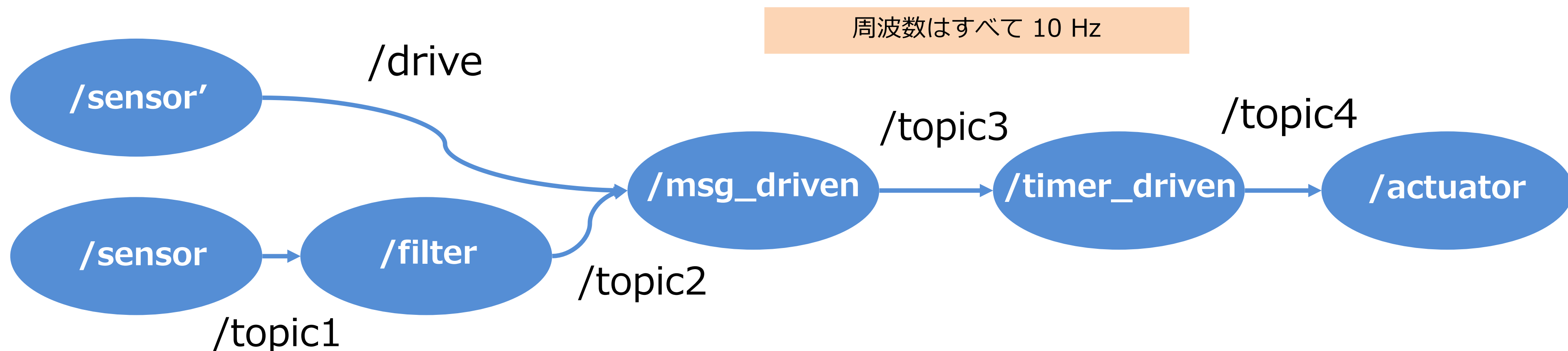
コールバックグラフ

End-to-Endレイテンシ
コールバック実行時間
ノード実行時間
user code (callback)
rclcpp
メッセージの流れ
RCL の通信レイテンシ
rcl
publish
rmw
DDS の通信レイテンシ
DDS

CARET の主要なトレースポイントと測定対象

ISP

■ ロボットのノード構成を単純化したアプリケーションを例に、CARET による測定方法を説明していきます



評価対象のサンプルアプリケーション

エグゼキュータ：MultiThreadedExecutor
各トピックの周波数は 9.8~10 Hz

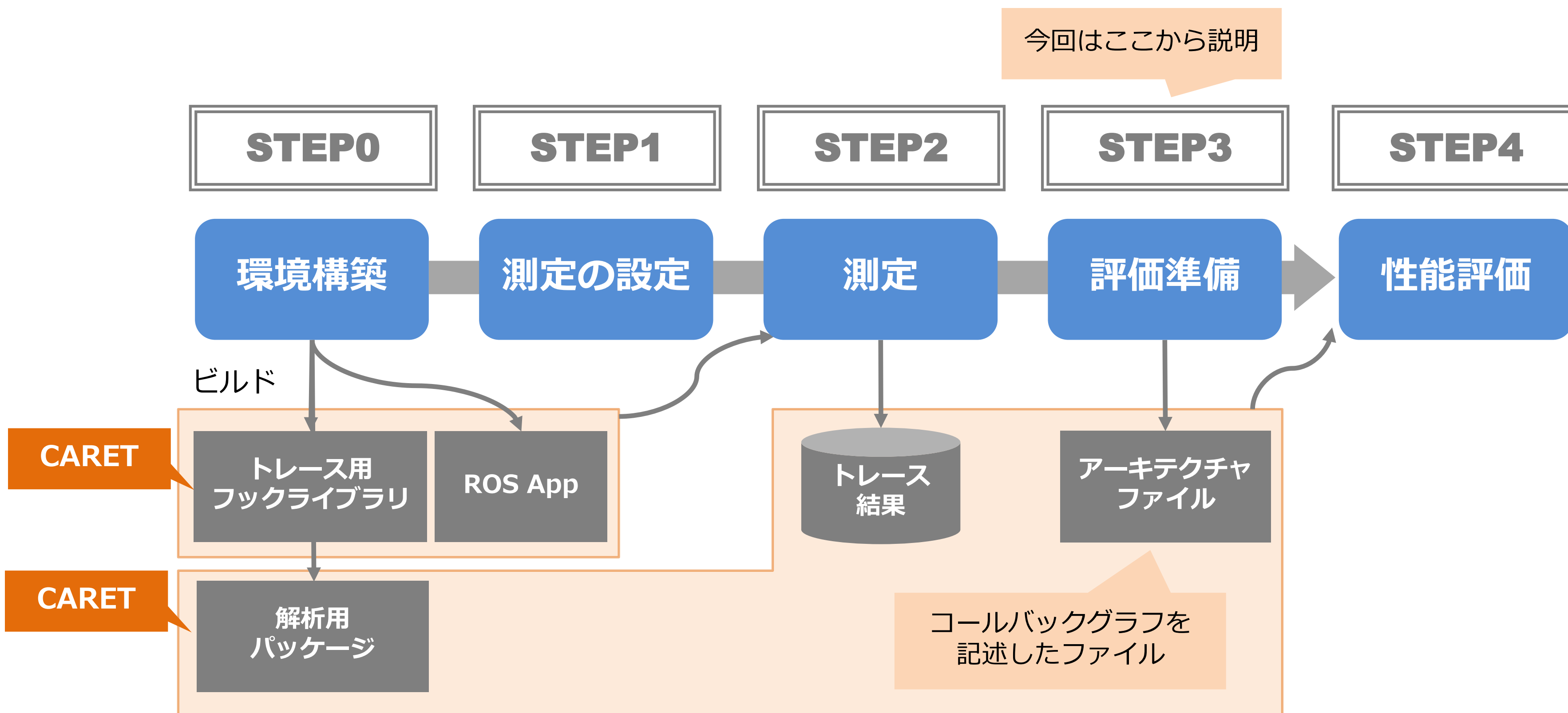
平均値上はおおむね動いている様子

「意図せず以下の実装となっていた」というシナリオで CARET で測定します。

QoS history size：10
/filter コールバックの実行時間はまれに 200 ms

CARET による評価の流れ

ISP



測定の設定

■ launch ファイルの修正

```
[.launch.py]
def generate_launch_description():
    return launch.LaunchDescription([
        Trace(
            session_name='my_trace_session',
            events_kernel=[],
            events_ust=['ros2*']
        ),
        launch_ros.actions.Node(...),
    ])
```

カーネルのトレースを無効化

ros2 trace と CARET のトレースを有効化

■ 環境変数の設定

共有ライブラリの探索で優先される。
フックによりトレースポイントの追加

```
$ export LD_PRELOAD=/path/to/libcaret.so
```

測定

```
$ ros2 launch caret_demo end_to_end_sample.launch.py
```

⇒ トレース結果が保存される。デフォルトでは~/.ros/tracing/my_trace_session

ISP

アーキテクチャファイルを作成する

コールバックグラフ構築に必要な情報を記述したファイル
トレース結果から雛形の自動生成が可能

[.yaml]

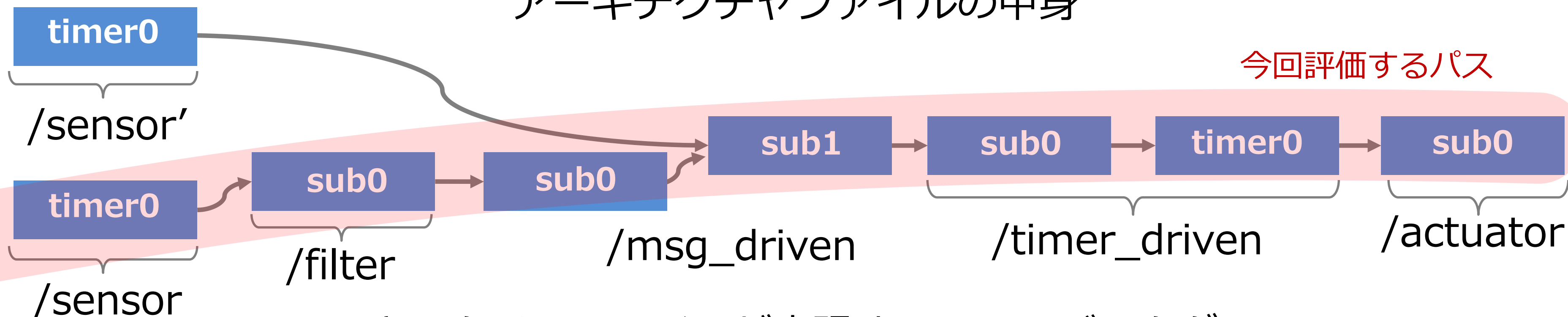
```yaml
...
- node_name: /filter
  callbacks:
  - callback_name: subscription_callback_0
    type: subscription_callback
    topic_name: /topic1
    symbol: Filter::{lambda()#1}
  publish:
  - topic_name: /topic2
    callback_name: subscription_callback_0
```

ノード情報
- ノード名

コールバック情報
- コールバック名
- パラメータ
- シンボル名

callback と publisher の紐付け

アーキテクチャファイルの中身



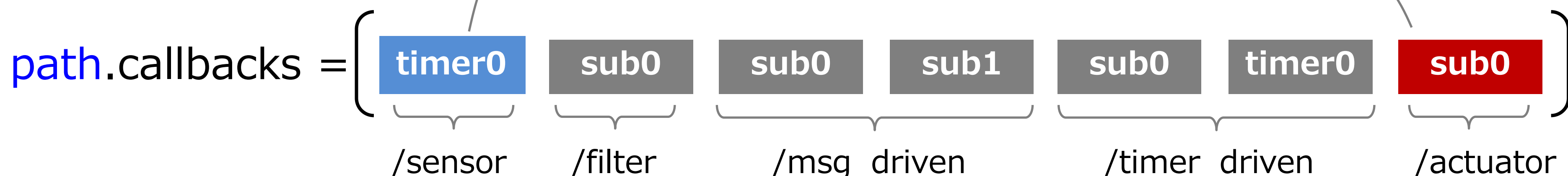
アーキテクチャファイルが表現するコールバックグラフ

評価対象のパスを指定する

- 評価対象のパスの始点と終点を指定し、パスの選択

```
import caret_analyze as caret

lttng = caret.Lttng('トレース結果のパス')
app = caret.Application('アーキテクチャファイルのパス', file_type='yaml', lttng)
path = app.search_paths(        ※ コールバック識別名 = ノード名/コールバック名
    '/sensor/timer_callback_0' ,'/actuator/subscription_callback_0' )[0]
```



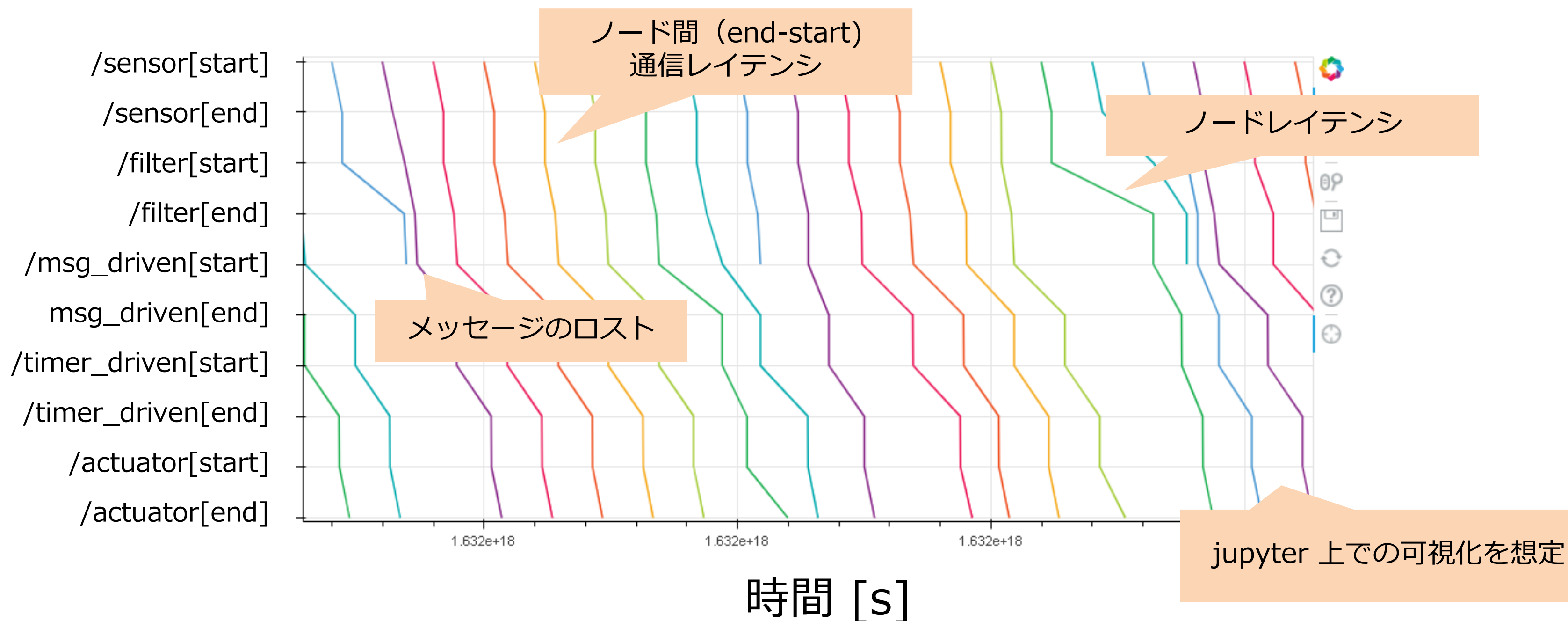
評価対象のコールバックチェーン

コールバックチェーン上のメッセージの流れを可視化

```
import caret_analyze.plot as caret_plot
caret_plot.message_flow(path, granularity='node')
```

[raw / callback / node] が指定可

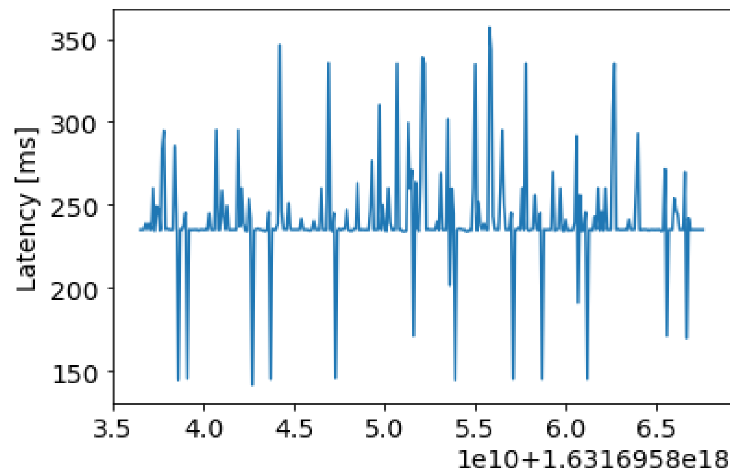
それぞれの線はメッセージの流れを示します。



/topic1 メッセージ数 315 個、その内 /actuator に到達したのは 290 個

■ End-to-Endレイテンシの時系列を可視化

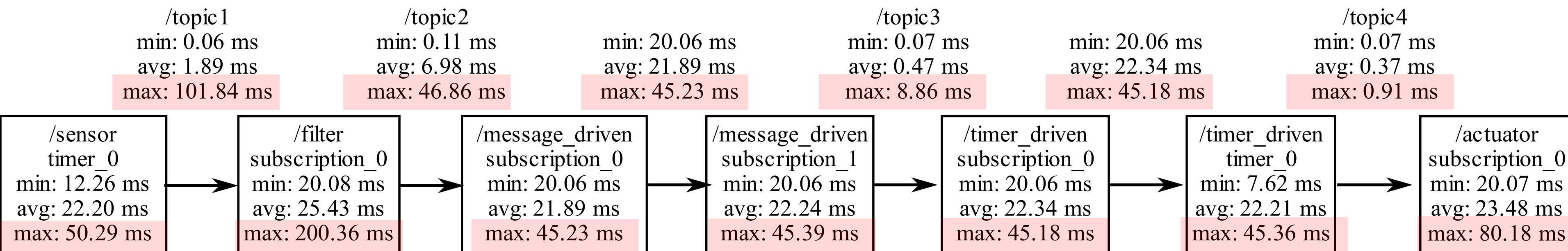
```
t, latency = path.to_timeseries()
plt.plot(t, latency * 1.0e-6)
```



実測：End-to-End レイテンシ：141 ~ 356 ms

■ コールバックチェーンのレイテンシを可視化

```
caret_plot.chain_latency(path, granularity='callback')
```



最大 End-toEnd レイテンシの推定値＝各最大レイテンシの総和 ≒ 750 ms

※ 最大値としては、楽観的（小さめ）な見積もり

⇒ CARET での測定により、 ノードやアプリケーションの評価が可能に

まとめ

ISP

- 動機
  - ROS2 アプリケーションの性能を評価したい

- ツールの紹介
  - 既存のツール
    - Linux のツール
    - ROS のツール
  - CARET
    - 詳細ドキュメント
      https://tier4.github.io/CARET_doc/

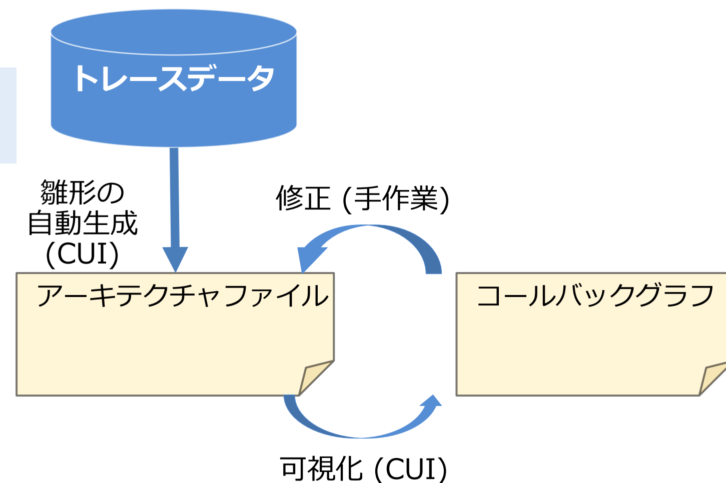
この成果は、国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の助成事業（JPNP16007）の結果得られたものです。

参考資料

- Performance Analysis and Tuning on Modern CPUs
- 詳解 システム・パフォーマンス
- Getting started with ROS 2 tracing

■ 雛形の自動作成

```
$ ros2 caret architecture -t trace_result –o architecture.yaml
```



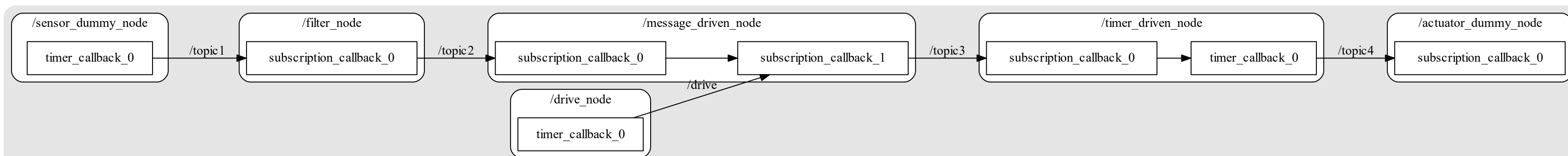
アーキテクチャファイル作成のフロー

■ 修正

- yaml ファイルを直接編集

■ コールバックグラフの可視化

```
$ ros2 caret callback_graph –a architecture.yaml -o callback_graph.svg
```



コールバックグラフを可視化した例

ISP

caret_plot.message_flow(path, granularity='raw')
# トレースポイントごとの結果を表示