Tracer ROS 2 avec `ros2_tracing`

Christophe Bédard

ROSCon Fr 2021
22 juin 2021

Laboratoire DORSAL
Polytechnique Montréal 🇨🇦

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Plan

1. Introduction
2. Contexte
3. Traçage & LTTng
4. `ros2_tracing`
5. Analyse
6. Démo
7. Conclusion
8. Questions

Introduction

- Robotique
  - Différents types d'applications
  - Jouets, applications commerciales, applications industrielles
  - Applications & systèmes critiques
- ROS 2
  - Nouvelles possibilités
  - Systèmes répartis
  - Contraintes temps-réel

Contexte

- Outils de débogage et diagnostiques
  - Débogage : GDB
  - Logs : ROS, printf()
  - Introspection : rqt_graph
  - Autres : diagnostic_aggregator, libstatistics_collector
- Problèmes avec l'observabilité
  - Effet de l'observateur
  - Éviter d'influencer le comportement de l'application
- Systèmes répartis
  - Comment analyser tout un système réparti?
- Temps réel
- Observer le déterminisme d'une application

Traçage

- But : récolter de l'information sur l'exécution d'une application
  - Information bas niveau
  - OS et application
- Utile quand les problèmes sont difficiles à reproduire
- Plusieurs traceurs avec différentes fonctionnalités
  - LTTng, perf, Ftrace, eBPF, DTrace, SystemTap, Event Tracing for Windows, etc.
- Principe (instrumentation statique)
  - Instrumente une application avec des points de trace
  - Configure le traceur, exécute l'application
  - Points de trace génèrent des événements (information)
  - Événements forment une trace
- On veut minimiser le surcoût d'utilisation!
  - Effet de l'observateur
  - Utiliser en production

LTTng

- lttng.org
- Traceur haute performance
  - Bas surcoût d'utilisation
- Linux seulement
- Instrumentation
  - Intégrée au noyau Linux
  - Ou à ajouter statiquement dans une application
- Traitement des données de la trace
  - En ligne (*live*)
  - Hors ligne

LTTng - exemple

```
$ lttng create ros2-session
$ lttng enable-event --kernel sched_switch
$ lttng enable-event --userspace ros2:rclcpp_publish
$ lttng enable-event --userspace ros2:*
$ lttng start
$ ros2 run pkg exe
$ lttng stop && lttng destroy
```

LTTng - exemple (2)

```
$ babeltrace ros2-session/
sched_switch: { cpu_id = 1 }, { prev_comm = "swapper/1", prev_tid = 0, prev_prio = 20,
    prev_state = ( "TASK_RUNNING" : container = 0 ), next_comm = "test_ping", next_tid =
    416160, next_prio = 20 }
ros2:callback_start: { cpu_id = 1 }, { callback = 0x541190, is_intra_process = 0 }
ros2:rclcpp_publish: { cpu_id = 1 }, { publisher_handle = 0x541A40, message = 0x5464F0 }
ros2:rcl_publish: { cpu_id = 1 }, { publisher_handle = 0x541A40, message = 0x5464F0 }
ros2:rmw_publish: { cpu_id = 1 }, { rmw_publisher_handle = 0x541AE0, message = 0x5464F0 }
ros2:callback_end: { cpu_id = 1 }, { callback = 0x541190 }
```

ros2_tracing

- gitlab.com/ros-tracing/ros2_tracing
- Collection d'outils
- Intégration étroite avec ROS 2
  - Pour encourager l'utilisation et l'adoption
  - Depuis ROS 2 Dashing (2019)
- Outils pour l'instrumentation du noyau de ROS 2 avec LTTng
  - `rclcpp`, `rcl`, `rmw`
- Outils pour configurer le traçage avec LTTng
  - Commande : `ros2 trace`
  - Action pour ROS 2 launch : `Trace`

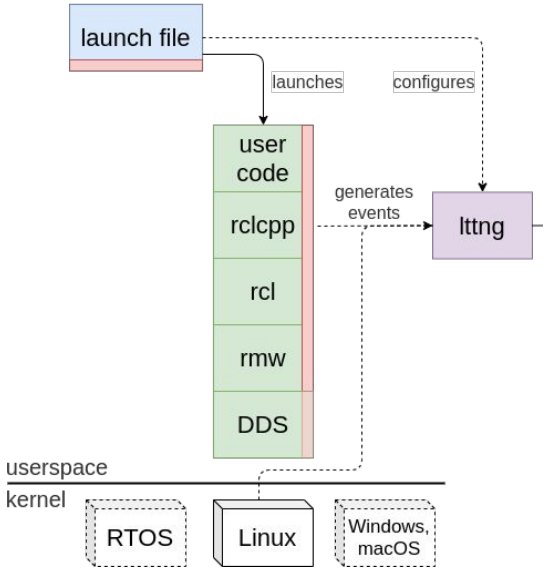

Figure 1. Instrumentation et principe d'utilisation.

Instrumentation

- Seulement sur Linux, pas incluse dans les binaires
  - Installer LTTng et (re)compiler ROS 2
- Instrumentation conçue pour supporter plusieurs traceurs
  - D'autres traceurs et/ou d'autres OS, éventuellement
  - `rclcpp`, `rcl`, `rmw`, etc. → `tracetools` → LTTng
- Principes de conception
  - Extraire de l'information à propos des niveaux d'abstraction
  - Mais les abstractions rendent la tâche plus difficile
- Temps réel
  - Applications ont généralement une phase d'initialisation non temps réel
  - On en profite pour récolter le plus d'information possible au début
  - Diminue le surcoût par la suite, en "régime permanent"

Instrumentation (2)

- Instances d'objets
  - Noeud, pub, sub, timer
- Événements
  - Exécution des callbacks (sub, timer)
  - Publication de message
  - Changement d'état d'un lifecycle node
  - Etc.
- Applicable à la plupart des niveaux d'abstraction
  - `rclcpp`, `rcl`, `rmw`
  - DDS (en cours avec Eclipse Cyclone DDS)

Instrumentation - exemple

- Noeud ping : un timer déclenche périodiquement la publication d'un message

```
ros2:rcl_node_init: { node_handle = 0x🚀, rmw_handle = 0x..., node_name = "test_ping" }
ros2:rcl_publisher_init: { publisher_handle = 0x🇨🇦, node_handle = 0x🚀, topic_name = "/ping", queue_depth = 10}

ros2:rcl_timer_init: { timer_handle = 0x🕛, period = 500000000 }
ros2:rclcpp_timer_callback_added: { timer_handle = 0x🕛, callback = 0x🤖 }
ros2:rclcpp_callback_register: { callback = 0x🤖, symbol = "std::_Bind<void (PingNode::*(PingNode*))()>" }

ros2:callback_start: { callback = 0x🤖, is_intra_process = 0 }
      ros2:rclcpp_publish: { publisher_handle = 0x🇨🇦, message = 0x🇫🇷 }
      ros2:rcl_publish: { publisher_handle = 0x🇨🇦, message = 0x🇫🇷 }
      ros2:rmw_publish: { rmw_publisher_handle = 0x..., message = 0x🇫🇷 }
ros2:callback_end: { callback = 0x🤖 }
```

Outils - commande `ros2 trace`

```
$ ros2 trace \
    --session-name ros2-session \
    --kernel sched_switch \
    --ust ros2:rclcpp_publish ros2:*
writing tracing session to: /home/chris/.ros/tracing/ros2-session
press enter to start...
press enter to stop...
stopping & destroying tracing session
```

Outils - action `Trace` pour ROS 2 launch

```
from launch import LaunchDescription
from launch_ros.actions import Node
from tracetools_launch.action import Trace
def generate_launch_description():
   return LaunchDescription([
       Trace(
           session_name='ros2-session',
           events_kernel=['sched_switch'],
           events_ust=['ros2:rclcpp_publish', 'ros2:*'],
       ),
       Node(
           package='pkg',
           executable='exe',
       ),
   ])
```

Évaluation du surcoût

- But : mesurer le surcoût d'utilisation dans un contexte de ROS 2
  - On s'intéresse au surcoût en latence
  - On s'attend à des petites valeurs
  - Outil : gitlab.com/ApexAI/performance_test
- Paramètres
  - Inter-processus : 1 pub → 1 sub
  - Publication : 100 - 2000 Hz
  - Messages : 1 - 256 Ko
  - Qualité de service : reliable
  - Eclipse Cyclone DDS
- Configuration
  - Ubuntu Server 20.04.2 avec PREEMPT_RT (5.4.3-rt1)
  - Intel i7-3770 @ 3.40GHz
  - SMT/Hyper-threading désactivé (4 coeurs, 1 fil/coeur)
  - Exécuter pendant 20 minutes, retirer les premières 5 secondes et utiliser la latence moyenne

Évaluation du surcoût - résultats

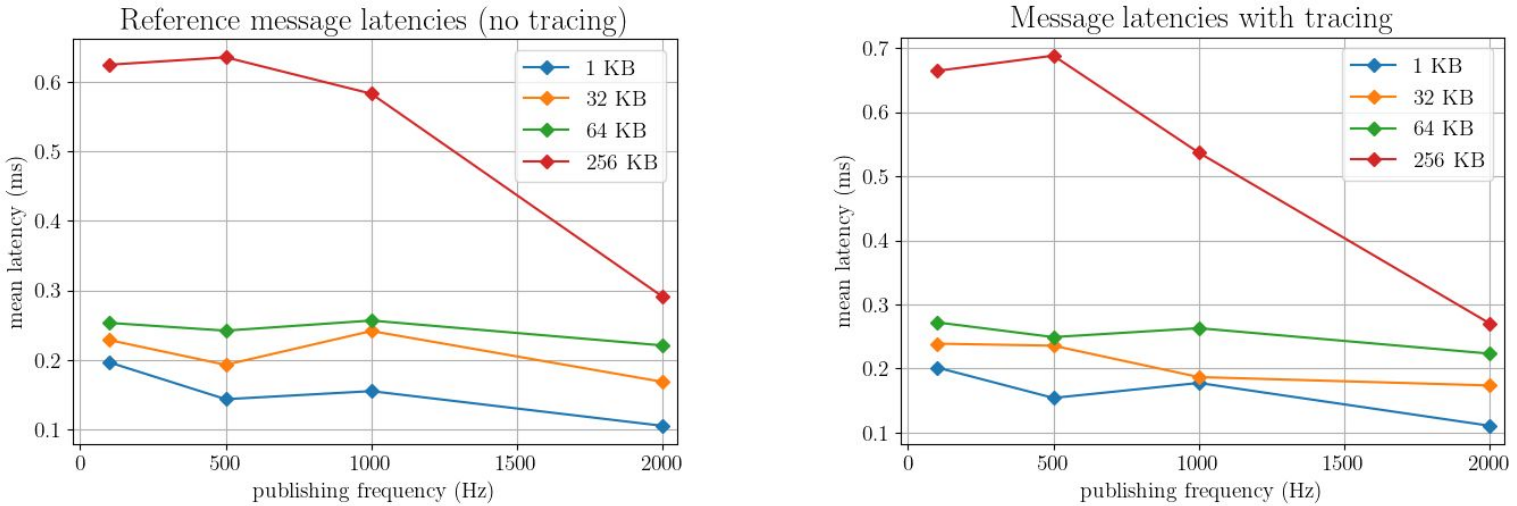

Figure 2. Résultats individuels.

Évaluation du surcoût - résultats (2)

- Difficile de conclure, mais semble encourageant
- Possiblement trop de variabilité au niveau de l'OS et du réseau

- Revoir la configuration
- Autres évaluations
  - Réparties
  - Utiliser les valeurs de médiane

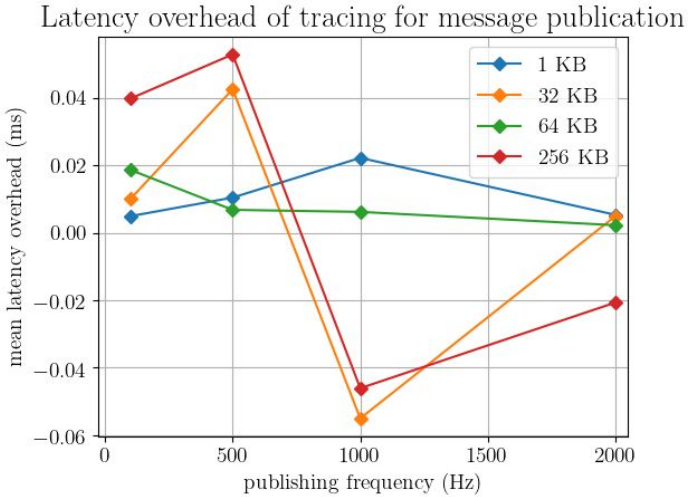

Figure 3. Résultats d'évaluation du surcoût.

Analyse

- Quelques outils pour analyser les traces produites par LTTng
  - babeltrace : babeltrace.org
  - Trace Compass : tracecompass.org
- `tracetools_analysis`
  - gitlab.com/ros-tracing/tracetools_analysis
  - But : analyse rapide des données
  - Outil simple en Python
  - Pré-traite les données brutes et produit des pandas DataFrame
  - Offre des fonctions simples pour analyser les DataFrames
  - Combiner avec un Jupyter Notebook
- Analyses plus complexes
  - **Corréler** les événements ROS 2 de la trace avec ceux du noyau Linux
  - **Analyser l'agrégation** des traces de plusieurs systèmes

Analyse - exemple

```
import tracetools_analysis; import bokeh
events = load_file('~/.ros/tracing/pingpong')                    # Lire la trace
handler = Ros2Handler.process(events)                            # (Pré-)traiter les données
data_util = Ros2DataModelUtil(handler.data)
callback_symbols = data_util.get_callback_symbols()              # Extraire les fonctions de callback
duration = bokeh.plotting.figure(...)
for obj, symbol in callback_symbols.items():                     # Pour chaque callback...
   owner_info = data_util.get_callback_owner_info(obj)
   if not owner_info or '/parameter_events' in owner_info:       #   Filtrer les subscriptions internes
       continue
   duration_df = data_util.get_callback_durations(obj)           #   Obtenir les données de durée
   duration.line(x='timestamp', y='duration', legend=str(symbol),#   Ajouter au graphique
                 source=bokeh.models.ColumnDataSource(duration_df))
bokeh.io.show(duration)                                          # Afficher le graphique final
```

Analyse - exemple (2)

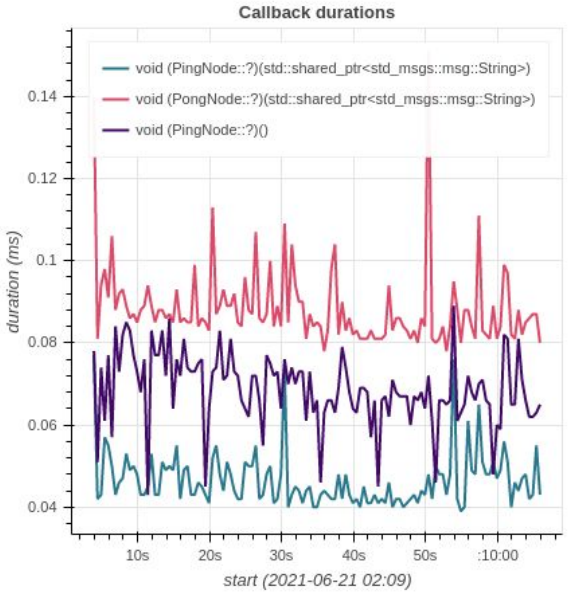

Figure 4. Analyse de la durée des callbacks.

Analyse - exemple (3)

- Analyse du chemin critique d'une requête wget
- Calcule les dépendances entre les fils d'exécution
- Seulement avec des données du noyau Linux
  - Appels systèmes bloquants

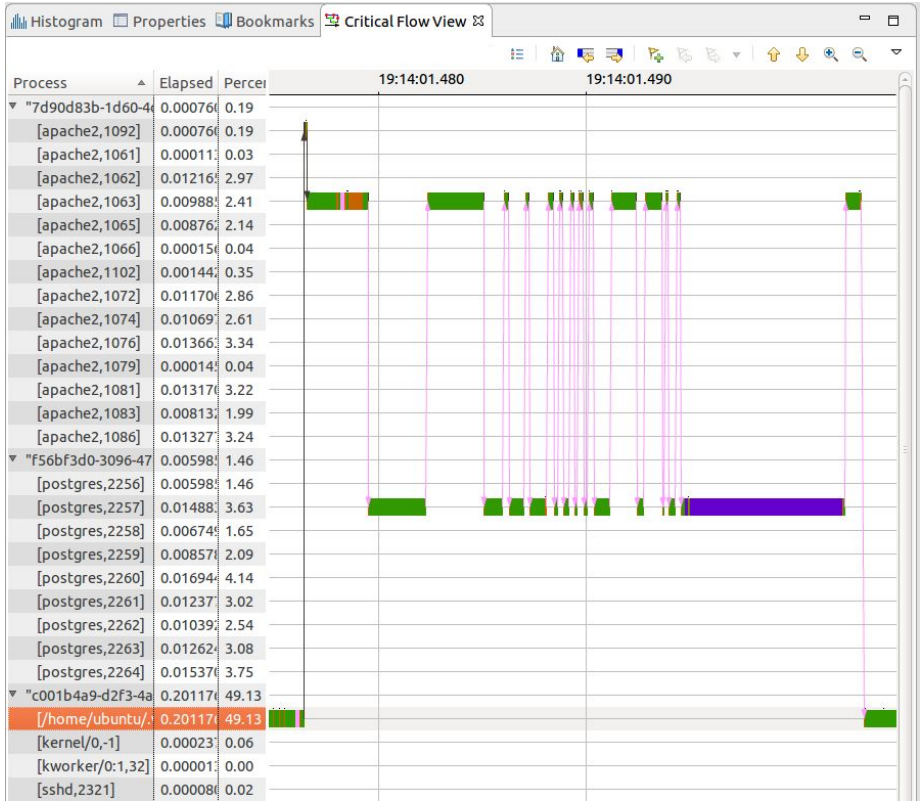

Figure 5. Analyse du chemin critique avec Trace Compass.

Démo

- ...avec ros2_control!
  - Instrumentation du controller manager
  - Extrait d'information à propos des controllers : `init()` et `update()`
  - Comparaison avec la publication de messages sur `/dynamic_joint_states`
- Instructions et code Python dans un Jupyter Notebook
  - gitlab.com/ros-tracing/tracetools_analysis/-/blob/add-basic-ros2-control-demo/tracetools_analysis/analysis/ros2_control_demo.ipynb

Démo - résultats

- Simple démo
- Beaucoup d'information, beaucoup de possibilités!

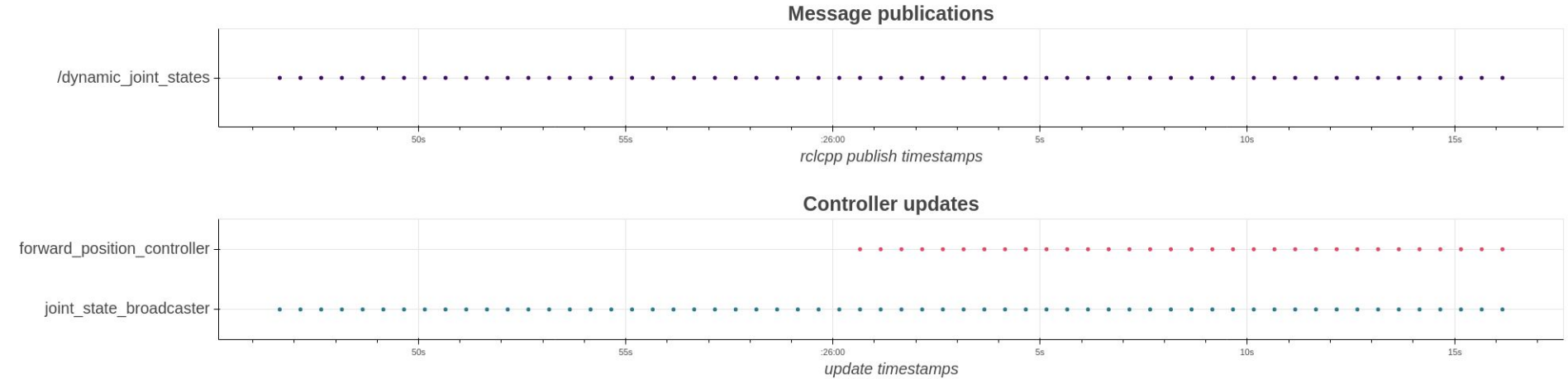

Figure 6. Résultats du démos.

Conclusion

- Traçage
  - Récolter de l'information bas niveau sur l'exécution d'une application
  - Bas surcoût d'utilisation
- `ros2_tracing`
  - Outils pour instrumenter le noyau de ROS 2
  - Outils pour configurer le traçage avec LTTng
- Analyse
  - Corréler les événements OS et ROS 2
  - Analyser l'agrégation des traces de plusieurs systèmes
- Futur
  - Inclure l'instrumentation par défaut dans les binaires Linux?
  - Instrumentation
    - Exécuteur, traitement interne des messages
    - DDS
  - Que voudriez-vous voir?!

Questions?

- github.com/christophebedard

- Liens importants
  - lttng.org
  - gitlab.com/ros-tracing/ros2_tracing
  - gitlab.com/ros-tracing/tracetools_analysis

Démo (2)

- Instrumentation pour ros2_control : `init()` et `update()`

```
if (controller.c->init(controller.info.name) == controller_interface::return_type::ERROR) {
  to.clear();
  RCLCPP_ERROR(
    get_logger(),
    "Could not initialize the controller named '%s'",
    controller.info.name.c_str());
  return nullptr;
}
executor_->add_node(controller.c->get_node());
to.emplace_back(controller);
TRACEPOINT(
  control_controller_init,
  static_cast<const void *>(controller.c.get()),
  controller.info.name.c_str());
```

```
controller_interface::return_type ControllerManager::update()
{
  std::vector<ControllerSpec> & rt_controller_list =
    rt_controllers_wrapper_.update_and_get_used_by_rt_list();

  auto ret = controller_interface::return_type::OK;
  for (auto loaded_controller : rt_controller_list) {
    // TODO(v-lopez) we could cache this information
    // https://github.com/ros-controls/ros2_control/issues/153
    if (is_controller_running(*loaded_controller.c)) {
      auto controller_ret = loaded_controller.c->update();
      TRACEPOINT(control_controller_update, static_cast<const void *>(loaded_controller.c.get()));
      if (controller_ret != controller_interface::return_type::OK) {
        ret = controller_ret;
      }
    }
  }

  // there are controllers to start/stop
  if (switch_params_.do_switch) {
    manage_switch();
  }

  return ret;
}
```

Démo (3)

- Launch file

```
def generate_launch_description():
    launchfile_path = os.path.join(
        get_package_share_directory('ros2_control_demo_bringup'),
        'launch',
        'rrbot_system_position_only.launch.py',
    )
    base_ros2_control_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([launchfile_path]),
        launch_arguments={ 'start_rviz': 'True' }.items(),
    )
    return LaunchDescription([
        Trace(
            session_name='ros2-control-demo',
            events_ust=[
                'ros2:*',
                'ros2:control_controller_init',
                'ros2:control_controller_update',
            ],
        ),
        base_ros2_control_launch,
    ])
```