[Proposal] An efficient way to handle topic messages by reducing invocations of subscription callback functions #4612

ohsawa1204 started this conversation in **Design**

ohsawa1204 on Apr 10 Collaborator edited

Introduction

High CPU time consumption is one of the critical issues that attract the interest of Autoware users. According to the breakdown of CPU time consumption per shared library shown by Intel VTune Profiler, we can see that some shared libraries that do not describe user logic consume a lot of CPU time. Perhaps more than 50% of the CPU time consumed by Autoware is caused by non-user logic. In other words, some anti-patterns in architecture or implementation in Autoware can increase CPU time consumption.

As you know, to reduce the CPU time consumption, it is not enough to reduce the CPU time consumed by the user logic described in the callback function. Fixing some anti-patterns is also critically effective to reduce CPU time consumption. This post is a proposal to fix one of the anti-patterns; overuse of callback functions.

Since Autoware follows a typical implementation shown in the ROS 2 tutorials, nodes in Autoware call a callback function upon receipt of each topic message. Calling a callback function costs trivial CPU time. For example, unintentionally calling a callback function for an unused message wastes CPU time. Besides, waking up a thread for a callback function causes CPU overhead. Such CPU overheads are small enough to be ignored for a small toy application, but it can become an issue for a large scale application like Autoware.

To tackle with such CPU overhead due to subscription callback functions, we'd like to propose another way to handle received topic messages.

Topic message processing in current Autoware

Topic messages are passed from a publishing node to a subscription node. If a topic message is subscribed, a dedicated callback function for the topic is called and then the message is processed. Here is an illustration of a typical node in Autoware:

Category

Design

Labels

None yet

4 participants

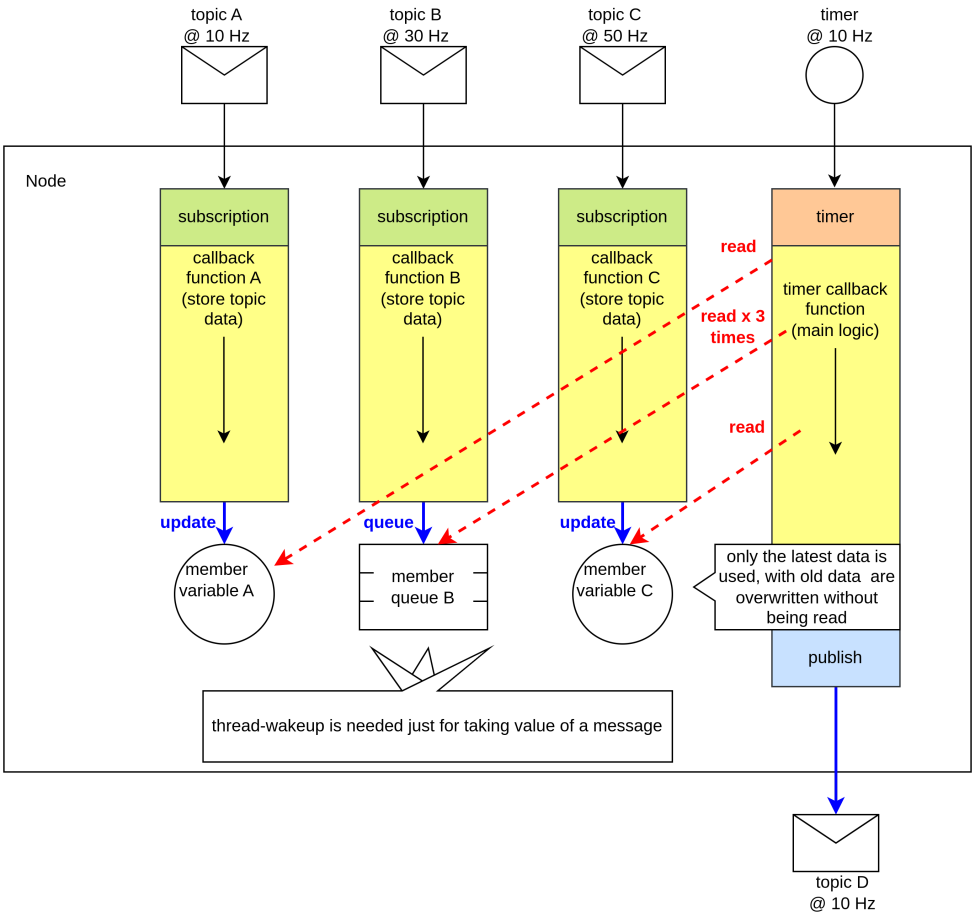

fig.1

The node receives three topics - topic A @ 10Hz, topic B @ 30Hz, and topic C @ 50Hz. Received data through the topics are passed to callback functions as its argument and just copied to member variables or queue of the node. When a timer callback function is invoked by 10Hz timer expiration, the data are read and processed by the timer callback function and then it publishes topic D.

We think there is a room for improvement from the point of CPU utilization.

- thread-wakeup is needed just for taking value of a message data, which is relatively expensive.
- member variable C is overwritten at 50Hz rate but only the last data is read. 4/5 of data are discarded with no meaning. Moreover, thread-wakeup is needed at every reception of topic C.

Our proposal

We propose an enhanced way of subscription.

- subscription callback function is not called when topic message is received
- timer callback function takes topic message data by itself

Here is an illustration which explains our proposal:

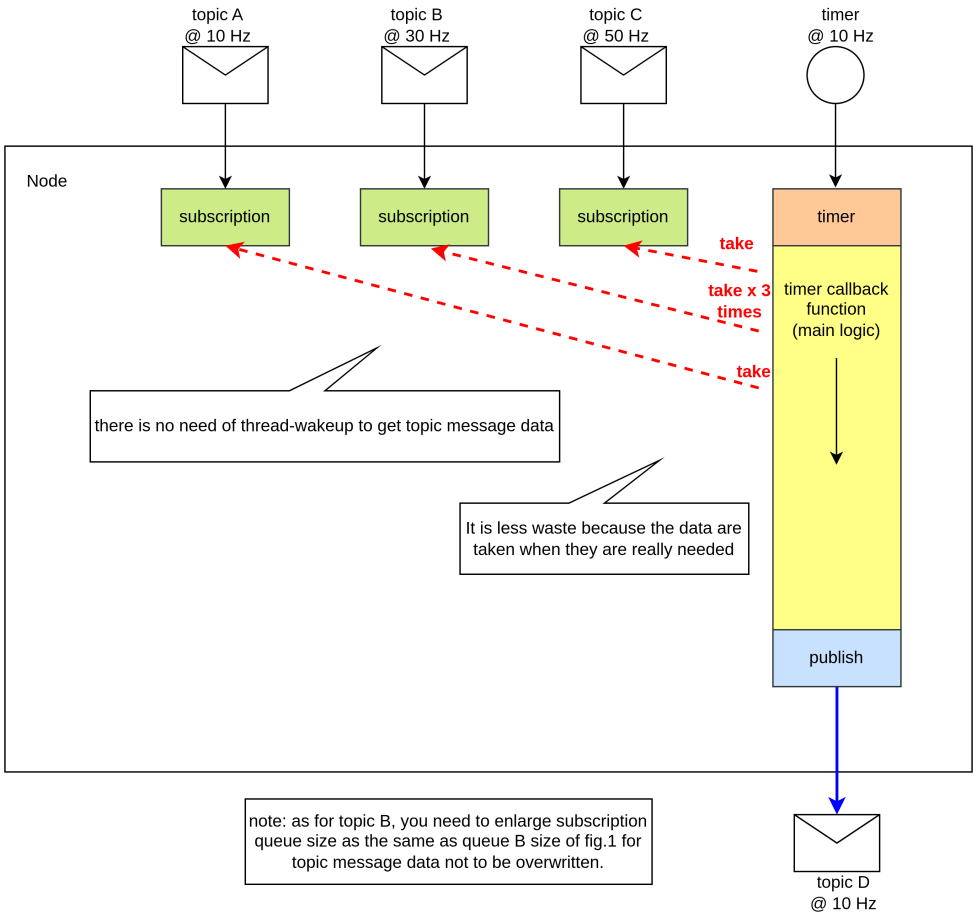

fig.2

You may not be familiar with such a way, but it can be seen in formal ROS2 examples. Please refer to [examples/rclcpp/wait_set/src/listener.cpp at rolling · ros2/examples] for example.

Expected achievement by the proposal

The benefits of our proposal are as follows:

- it can reduce invocations of subscription callback functions which we think relatively expensive
- there is no need to take topic message data which are not read
- there is no need of exclusive lock between subscription callback thread and timer callback thread

Source code change

To do this, we need to modify source code as below:

1. do not call subscription callback function when topic message is received

- create a callback group with specifying `false` as the second argument of `create_callback_group` (L2)
- set the created callback group to `callback_group` member of SubscriptionOptions (L4)
- create a subscription with specifying the SubscripionOptions as the fourth argument of `create_subscription` (L9)

```
  rclcpp::CallbackGroup::SharedPtr cb_group_noexec = this->
      rclcpp::CallbackGroupType::MutuallyExclusive, false);
  auto subscription_options = rclcpp::SubscriptionOptions();
  subscription_options.callback_group = cb_group_noexec;
  rclcpp::QoS qos(rclcpp::KeepLast(10));
  if (use_transient_local) {
    qos = qos.transient_local();
  }
  sub_ = create_subscription<std_msgs::msg::String>("chatter", qos
```

2. take messages from timer callback function

- call `take` method of subscription (L3)

```
std_msgs::msg::String msg;
rclcpp::MessageInfo msg_info;
if (sub_->take(msg, msg_info)) {
  RCLCPP_INFO(this->get_logger(), "Catch message");
```

That's all basically. **Note that there is no need to change algorithm or logic of user code.** All we need is just to change the way of taking topic message data. They are quite straightforward changes, therefore low risk of degradation is expected. In fact, we have just applied the change to a part of Autoware. Please refer to feat(tier4_autoware_utils, obstacle_cruise): change to read topic by polling by yuki-takagi-66 · Pull Request #6702 · autowarefoundation/autoware.universe for a better understanding of the change.

some additions about the change

- If the frequency of subscription callback is larger than that of timer callback (e.g. 50Hz vs 10Hz) and all topic messages need to be read, it is needed to change size of the queue of the subscription adequately and all messages must be taken in timer callback function calling `take` method within loop.
- If reception time is obtained by such as `rclcpp::Time::now` in subscription callback function, keep in mind that reception time changes by the change of our proposal
  - timestamp of received message may be used instead in that case
  - if this is not acceptable, it is considerable not to apply the change to the subscription callback
- `take` method of subscription can be used for inter process message passing, but can not be used for intra process message passing
  - `take_data` and `execute` method can be used for intra process message passing instead
- taking message data from subscription queue by `take` method is irreversible, therefore once data is taken, it can not be returned to the queue
  - there is a way to verify a message is on subscription queue by using `rclcpp::WaitSet`
  - a message stays remaining in that case

Performance measurement

We had some experiments to verify improvements by the change of our proposal. We applied the change to behavior planning, motion planning, control and some other nodes in Autoware and ran them on real vehicle. CPU utilization of the changed nodes decreased about 20%. We used top command, perf command, and Intel VTune Profiler to measure, and almost similar result was obtained with any of those. Of course, the degree of the decrease of CPU utilization depends on how many subscription callback invocations decrease. We chose nodes which had many subscription callbacks so that the effective performance improvement were expected. If the change is applied to a node which has a few callbacks, it is not expected to achieve such an improvement like this, but at least some improvement will be obtained.

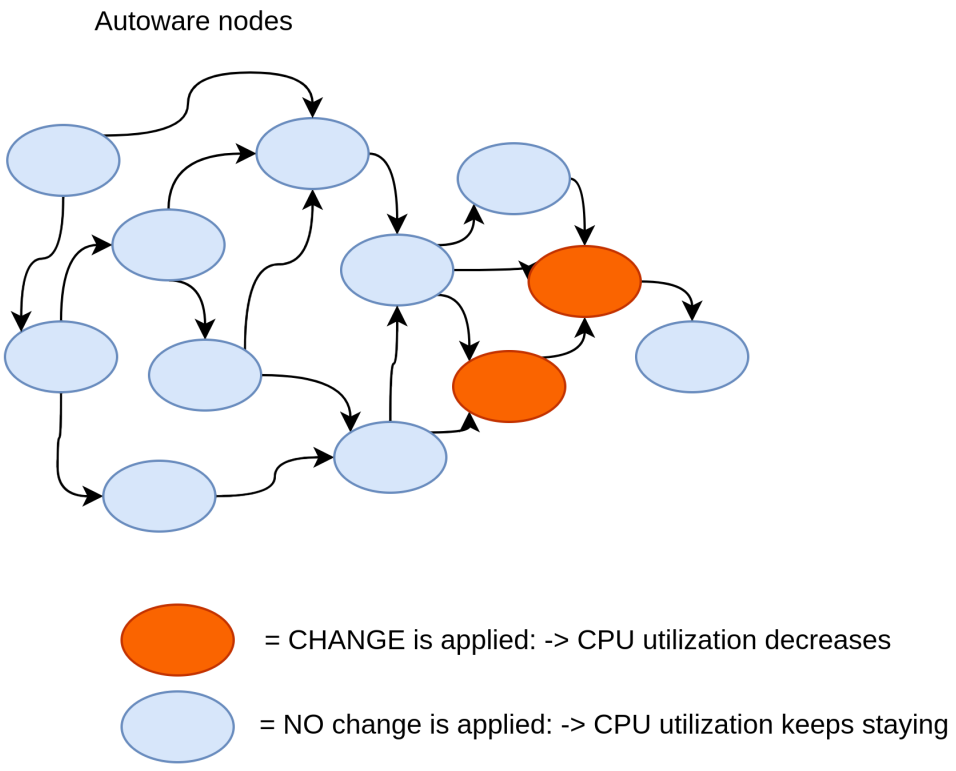

fig.3

Note that nodes to which the change is applied and those to which the change is not applied can be mixed together because the interface of message passing between nodes is not changed. Therefore we can apply the change each by each.

Here is an experiment result obtained using Intel VTune Profiler with measurement duration 1 minute on a real vehicle on which Autoware was running. The measurement was done with the vehicle stopped and driving route set from ego position to a goal. Generally speaking, measurement data depends on surroundings and situations, so take these data just as reference. Source code repositories used for the experiment are here. Those source code are written for only the experiment, but not for merging upstream.
GitHub - ohsawa1204/autoware.universe at evaluate_reduction_callback
GitHub - takam5f2/tier4_ad_api_adaptor at perf_callback_reduction

measurement target	total running time of 16 CPU cores AFTER change	total running time of 16 CPU cores BEFORE change	reduction rate
system-wide	588.85 sec	644.09 sec	8.58%
behavior planning	31.15 sec	39.67 sec	21.46%
motion planning	11.40 sec	14.40 sec	20.81%
control	28.75 sec	34.55 sec	16.80%
scenario planning (scenario selector)	0.483 sec	0.841 sec	42.6%
topic state monitor x 12	3.79 sec	4.13 sec	8.23%
awapi_awiv_adapter	2.28 sec	4.90 sec	53.5%

We observed the similar improvement with the Planning simulation tutorial.

Summary

- we propose an efficient way of taking a topic message data
- some degree of decrease of CPU utilization is expected
- the change is straightforward and then row risk of degradation is expected
- the change does not affect interface of message passing between nodes

Because of benefits above, we want to start apply the changes in Autoware.

↑ 3 👍 8

4 comments · 4 replies

Oldest | Newest | Top

TakaHoribe on Apr 18 Maintainer

@ohsawa1204 Thank you for your proposal. This approach is excellent because it does not require any changes to the algorithm and it is applicable to all ROS nodes. The performance improvement report `CPU utilization of the changed nodes decreased about 20%` is also outstanding.

There is just one point to note: this method is not commonly used in ROS2 notation, so it will be necessary to write appropriate guidelines in the documentation. However, with those in place, I believe there will be no big issues in this approach.

↑ 1

1 reply

ohsawa1204 on Apr 18 Collaborator Author

@TakaHoribe
Thank you very much for your comment.
I am writing a guideline for this topic and will post it to Autoware Documentation when it is finished.

ohsawa1204 on Apr 24 Collaborator Author

I added experiment result just for reference in `Performance measurement` section.

↑ 2 0 replies

ZhenshengLee on Jun 28 edited

The use of waitset-polling-subscription is of great value not only about cpu usage decrement, but also for realtime performance which is essencial for a av system like autoware.

This mechanism was developed as APEX_OS_POLLING_SUBSCRIPTION communicator, and was published(but not opensourced) in roscon2021 realtime workshop, https://www.apex.ai/_files/ugd/984e93_54790d76c0574748901a425555320b8a.pdf .

Hope the material helps the community and let us improve the waitset in rclcpp.

↑ 1 👍 2 0 replies

ZhenshengLee 3 weeks ago

> 2. take messages from timer callback function

@ohsawa1204

As the ros2 docs described, the timer callback created by rclcpp::timer is a kind of callback as well as subscription callback, so the timer is actually scheduled by executor.

Then as all nodes in autoware.universe are coded as component node, they are all managed nodes by ros2 launcher, so the launcher may launch the nodes with one kind of executor(ste, mte, sste etc.)

Theorytically, as there is business(algorithms) running in timer callback, it can cost too much time, and can block other timer callbacks in single threaded executor thread, also in multi-threaded executor threads without callback group settings.

So, if you really want to elimilate the extra threads in ros2 system, use waitset is not enough, we may not use executor, so the timer callback should changes to normal threads or cpp STL timer.

What's your idea?

EXTRA: the executor cannot be removed because there are other feature like service/actions/parameter service that depends on it, but if we change the timercallback to ordinary threads, the single threaded executor is definitly enough, so the number of thread are reducing.

↑ 1 ❤️ 1 3 replies

felixf4xu 3 weeks ago

> also in multi-threaded executor threads without callback group settings.

intereseting, I would like to know more about the `multi-threaded executor` and the `callback group settings`

ZhenshengLee 3 weeks ago

Hi, **@felixf4xu**

the basic concept of executor is documented here, https://docs.ros.org/en/humble/Concepts/Intermediate/About-Executors.html#types-of-executors

the latest news of executor is here in roscon2024 which describes some best practices, https://roscon.ros.org/2024/ search executor(slides are available soon)

callback group settings you can check the sample here, https://github.com/ros2/examples/tree/rolling/rclcpp/executors/cbg_executor
https://github.com/ros2/examples/tree/rolling/rclcpp/executors/multithreaded_executor

if you work in China, feel free to contact me through wechat: zhensheng_li

ohsawa1204 3 weeks ago Collaborator Author

@ZhenshengLee

Thank you very much for your comment!
It seems to me that your idea would work most efficiently with a node that only runs the timer callback and the subscription topics are taken with take() by that timer callback. I think it is a good idea.
In practice, I think it would be more realistic to run with a single threaded executor whenever possible, and to use take() to get the subscription topics whenever possible. I hope to make such modifications to Autoware.