ROS 네트워크의 정확한 모듈 수행 시간 분석 기법

한종우○ 이세환 이승수 조영은 이창건*
서울대학교 컴퓨터공학부
spiraline@snu.ac.kr, honestlee213@snu.ac.kr, sslee@rubis.snu.ac.kr, yecho@rubis.snu.ac.kr, cglee@snu.ac.kr

A Precise module execution time measurement of ROS network

Jongwoo Han○ Sehwan Lee Seungsu Lee Youngeun Cho Chang-Gun Lee*
Department of Computer Science and Engineering, Seoul National University

요 약

ROS(Robot Operating System)는 가장 널리 사용되는 로봇 제어 소프트웨어 프레임워크이다. 자율주행 등 ROS를 이용한 정밀 제어를 위해서는 수행 시간 측정이 필수적이다. 일반적으로 많이 쓰이는 리눅스의 time 명령어와 ROS의 rostopic 명령어는 ROS 네트워크 구조에 대한 고려가 부재하여 통신/유휴 시간 등으로 실제 수행 시간과는 상당한 차이를 보인다. 본 연구에서는 ROS 네트워크의 구독자/발행자 구조를 바탕으로 실제 수행시간만을 정확히 측정할 수 있는 방법을 제안한다. 나아가 이 과정에서 개발자가 만든 응용프로그램의 소스 코드 수정 없이 구현하여 편의성과 안정성을 확보하였으며, 제안한 기법이 기존의 방법에 비해 10배 이상 오차를 줄이는 유의미한 성능 개선이 있음을 확인했다.

1. 서론

ROS(Robot Operating System)는 로봇 제어를 위한 오픈소스 소프트웨어로 간단한 로봇부터 자율주행 자동차까지 응용가능한 범위가 매우 넓고, 쉽게 서비스를 개발할 수 있는 등 접근성이 높아 로봇 제어에서 가장 널리 사용되고 있는 프레임워크다[1][2].

자율주행 등 ROS를 이용한 정밀 제어를 위해서는 정확한 수행 시간 측정이 필수적이다. 구체적으로는 ROS로 구현한 시스템이 정상적으로 작동하지 않는 상황, 혹은 더욱 효율적인 시스템으로 최적화하기 위한 상황에서는 노드(Node) 단위가 아닌 함수 단위의 보다 자세한 수행 시간 분석이 필요하다.

그러나 이를 위해 사용되는 리눅스의 time 명령어나 ROS의 rostopic 명령어는 함수 단위의 측정이 불가능하거나, 통신/유휴 시간을 함께 측정하는 등 실제 수행 시간만을 측정하지 못한다. 이는 ROS 네트워크 구조에 대한 고려의 부재에 기인한다.

따라서 본 논문에서는 ROS 네트워크 구조 분석을 기반으로 한 분석 기법을 제안한다. 구체적으로, ROS 네트워크의 구독자가 수행하는 함수는 큐(Queue)에 쌓인 뒤 스피너가 이를 하나씩 꺼내며 수행한다.

제안하는 기법은 스피너 내부에서 각각의 콜백 함수만의 수행 시간만을 측정하여 스피너 외부에서 측정하는 기존의 방법보다 정확한 분석이 가능하다.

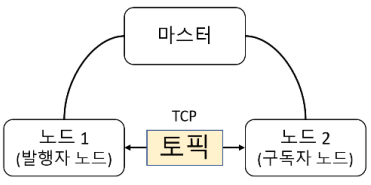
제안하는 기법을 통해 모듈 수행 시간을 (1) 개발자가 만든 응용프로그램의 소스 코드를 수정하지 않고, (2) 콜백 함수 단위에서 분석할 수 있다.

본 논문의 2장에서는 ROS 네트워크 구조에 대한 이론적 배경을 언급한다. 3장에서는 언급한 ROS 네트워크 구조를 바탕으로 새로이 설계한 분석 기법을 소개하고, 기존 방법과 비교하여 개선된 점을 명확히 한다. 4장에서는 본 연구의 결론을 정리한다.

2. 이론적 배경

본 장에서는 ROS 콜백 함수의 수행 시간 측정을 위한 분석 기법을 소개하기 전 ROS 네트워크 구조에 대해 이해가 필요한 내용을 소개한다. 2.1에서는 ROS의 토픽 기반 통신 구조에 대해 다루고, 2.2에서는 콜백 함수가 실행되는 과정에 대해 자세히 알아본다.

2.1 토픽 기반 ROS 통신 구조



[그림 1] 토픽 기반 ROS 통신 구조

ROS는 노드라고 불리는 단위로 기능을 수행하며

---

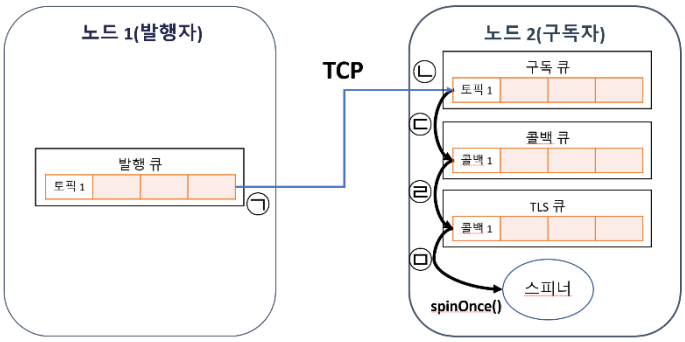
* 교신저자

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW컴퓨팅산업원천 기술개발사업(SW스타랩)의 연구결과로 수행되었음 (IITP-2015-0-00209)
이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다

전체 노드들은 마스터(Master)에 의해 관리된다.

본 논문의 모든 내용은 노드 간 통신 중 가장 많이 사용되는 토픽 기반의 통신구조에 대해 다룰 것이며 그 구조는 **[그림 1]**과 같다[3]. 토픽 기반 통신은 마스터의 주관 하에 메시지를 생성하는 발행자(Publisher) 노드와 메시지를 받아 이에 대응하는 콜백을 제공하는 구독자(Subscriber) 노드 사이에 TCP 통신을 통해 이루어진다[4].

2.2 콜백 함수의 동작



[그림 2] 콜백 함수가 실행되는 과정

구체적으로 콜백 함수가 실행되는 과정은 **[그림 2]**와 같다. 기본적으로 각 ROS 노드는 고유의 구독 큐, 콜백 큐, 발행 큐를 가진다. TCP 통신을 위한 사전 준비를 마치면 ㉠ 노드 1에 해당하는 발행자 노드는 발행 큐에 담긴 토픽 1을 TCP를 이용하여 발행한다. ㉡ 노드 2에 해당하는 구독자 노드가 토픽을 구독하게 되면 구독 큐에 해당 토픽이 들어가며 ㉢ 동시에 콜백 큐에는 대응하는 콜백 함수가 들어간다[5].

콜백 처리는 스피너에 의해 이루어진다. ㉣ 스피너가 동작 시 콜백 큐에 담겨 있는 콜백 정보들을 TLS 큐에 전달하고 콜백 큐는 비워지게 된다. 이후 스피너는 spinOnce()를 반복하여 실행하는데 spinOnce() 내에서 ㉤ TLS 큐에 담긴 콜백 함수들을 전부 비워질 때까지 하나씩 빼내며 실행한다. 따라서 TLS 큐 내부의 함수들이 실행되는 시간이 콜백 함수만의 순수한 수행 시간이다.

3. 분석 기법 제안

2.2에서 살펴본 콜백 함수의 실행 과정에 따르면 콜백 함수의 실제 수행 시간을 측정하기 위해서는 TLS 큐에서의 각 함수의 수행 시간을 측정하여야 한다.

이 때 개발자 코드를 직접 수정하여 함수의 수행 시간을 측정하는 것은 측정하고자 하는 모든 함수에 개발자가 직접 타임스탬프 출력 코드를 작성해야 한다는 편의성의 문제, 그리고 큰 프로젝트의 경우 코드의 삽입이 다른 모듈에 영향을 줄 수 있다는 안정성의 문제가 있다. 따라서 제안하는 기법은 개발자 코드의 수정이 없어야 한다.

본 장에서는 개발자 코드를 수정하지 않고 수행 시간을 분석할 수 있는 기존의 방법을 소개한다. 또한 본 논문에서 제안하는 분석 기법이 기존의 방법에 비해 가지는 장점에 대해 서술하고 성능 비교 실험을 통해 이를 검증한다.

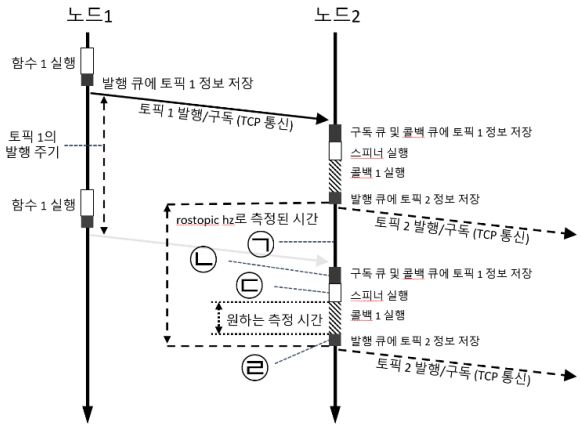
3.1과 3.2에서는 리눅스 내장 time 명령어와 ROS 내장 rostopic 명령어를 다루고, 3.3에서는 본 논문에서 제안하는 분석 기법에 대해 설명하며 기존의 방법에 비해 가지는 장점에 대해 서술한다. 3.4에서는 성능 비교 실험을 바탕으로 이를 검증한다.

3.1 리눅스 내장 time 명령어

프로그램을 time 명령어와 함께 실행하면 전체 수행 시간과 프로그램이 사용자 모드와 커널 모드에서 실행된 시간이 측정된다. 하지만 time 명령이 실행된 시점부터 쓰레드(Thread)가 종료된 시점까지 측정되기 때문에 원하는 시간만큼의 측정이 불가능하다[6].

또한 측정 단위가 CPU 내부의 쓰레드이기 때문에 쓰레드 내에서 실행되는 여러 함수 중 우리가 원하는 함수만의 수행 시간만을 측정할 수 없다. 특히 함수 단위의 측정이 아니기 때문에 루프를 이용하여 함수를 반복적으로 실행하는 ROS의 구조 상 time 명령어로는 한번의 수행(Iteration)에 대한 측정이 불가능하다. 본 논문에서 제안하는 기법은 함수 단위의 측정이 가능하기 때문에 수행 시간 분석에 더 적합하다고 할 수 있다.

3.2 ROS 내장 rostopic 명령어



[그림 3] rostopic hz와 제안하는 기법의 측정 방법

rostopic은 ROS에서 제공하는 명령어 라인 도구(Command Line Tool)다. 이 중 **rostopic hz** 명령어는 ROS가 발행하는 토픽의 발행 주기를 출력한다[3].

이 때 측정하는 주기는 **[그림 3]**에서 구독자 노드에 해당하는 노드 2가 토픽 2를 발행하는 주기에 해당한다. 콜백 1은 발행자 노드인 노드 1이 발행한 토픽 1을 성공적으로 구독하는 경우 실행되기 때문에 이상적인 경우 **rostopic hz**가 측정하는 주기는 토픽 1의 발행 주

기와 동일하다.

그러나 이 경우 큐에 정보를 저장하는 시간(㉡, ㉣), 스피너를 실행하는 시간(㉢) 등 부가적인 작업에 대한 시간이 포함된다. 또한 콜백 함수의 수행 시간이 토픽 1의 발행 주기보다 짧은 경우 노드의 유휴(Idle) 시간(㉠)이 함께 측정된다. 따라서 콜백 함수의 수행 시간을 측정하는 데 오차가 발생한다.

3.3 제안하는 기법

```
Algorithm 1: revised spinOnce()
Result: Execution time for callback function
create list TS;
while TLS Queue is not empty do
    dequeue;
    t_s ← current time
    run callback funtion;
    t_e ← current time
    TS.push(t_e − t_s);
end
return TS;
```

[그림 4] 제안하는 기법의 의사코드

2.2에서 진술한 바와 같이 구독자 노드 내의 스피너는 spinOnce()를 호출하여 콜백 함수를 하나씩 큐에서 꺼내며 실행한다. 이 과정은 ROS의 기본 라이브러리에 반복문을 이용하여 구현되어 있다. 제안하는 기법은 **[그림 4]**와 같이 이 반복문 안에서 콜백 함수의 실행 앞뒤로 타임스탬프를 기록한다. 이후 두 타임스탬프의 차를 계산하여 콜백 함수가 수행된 시간을 얻는다.

제안하는 기법은 스피너 내부 코드를 수정하기 때문에 스피너 자체의 실행 시간을 무시할 수 있다. (**[그림 3]**의 ㉢) 또한 **[그림 4]**의 dequeue와 같이 TLS 큐에 대한 작업을 제외한 오직 콜백 함수를 실행하는 시간만을 측정하기 때문에 큐에 정보를 저장하거나 빼내는 시간을 무시할 수 있다. (**[그림 3]**의 ㉡, ㉣) 또한 TLS 큐는 구독자 노드가 사용하기 때문에 발행자 노드에 의존하는 유휴 시간을 무시할 수 있다(**[그림 3]**의 ㉠). 따라서 리눅스의 time 명령어가 반복문 내부의 함수 각각의 수행 시간을 측정하지 못한다는 단점을 보완하고, ROS의 rostopic 명령어가 유휴 시간 등을 함께 측정하여 부정확한 수행 시간을 도출한다는 단점을 보완하여, 순수하게 콜백 함수만의 수행 시간만을 측정할 수 있다.

또한 ROS 라이브러리만을 수정했기 때문에 개발자의 코드에 의존하지 않고 범용적으로 사용이 가능한 기법이라고 할 수 있다.

3.4 성능 비교 실험

본 논문에서 제시하는 분석 기법과 **rostopic hz**의 성능 비교를 위해 **[그림 3]**의 상황을 재현하였다. 먼저 토픽 1을 10Hz로 발행하는 노드 1과 토픽 1을 구독하는 노드 2를 구현하였다.

토픽 1에 대한 콜백 1에서는 토픽 2가 발행되며 여러 작업이 수행되는 실제 상황을 재현하기 위해 함수 내에 반복문으로 출력 함수를 수행하게 하였다. 이후 **rostopic hz**로 토픽 2의 주기를 측정하였으며, 제안하는 기법으로는 콜백 함수의 수행 시간을 측정하였다. 이 과정을 1000회 반복하여 평균을 구하였다.

[표 1] rostpic hz와 제안하는 기법의 측정 결과

측정 방법	rostopic hz	제안하는 기법	비율
수행 시간	0.10 s	0.0093 s	9.3%

[표 1]과 같이 발행자 노드의 발행 주기보다 작업의 수행 시간이 짧은 경우에도 **rostopic hz**는 발행 주기와 동일한 0.10s (10Hz)가 측정되는 것을 확인할 수 있다. 이는 제안하는 기법이 측정한 0.0093s에 비해 10배 이상 높은 수치인데 이를 통해 **rotopic hz**는 통신/유휴 시간 등 오차가 되는 시간을 실제 수행 시간의 10배 이상 측정했다는 것을 확인할 수 있다.

따라서 기존의 방법으로는 콜백 함수만의 수행 시간을 측정하는 것이 불가능하며, 제안하는 기법으로 이를 수행할 수 있음을 확인할 수 있다.

4. 결론

본 논문을 통하여 ROS 네트워크의 각 모듈의 수행 시간을 정확히 측정하는 분석 기법을 제안하였다. 제안하는 기법은 개발자 코드의 수정 없이 라이브러리 형태로 사용하는 것이 가능하다. 또한 이 기법을 통해서 기존에는 부정확했던 함수 단위의 수행 시간 측정을 보다 정확하게 할 수 있다.

참고문헌

[1] S. Kato, et al., “An Open Approach to Autonomous Vehicles”, *IEEE Micro 35.6*, pp. 60-68., 2015

[2] S. Kato, et al., “Autoware on Board: Enabling Autonomous Vehicles with Embedded System”, *9th International Conference on Cyber-Physical Systems*, pp. 287-296., 2018

[3] J. Kerr, K. Nickels, “Robot Operating Systems: Bridging the Gap between Human and Robot”, *Southeastern Symposium on System Theory*, pp. 99-104., 2012

[4] M. Quigley, et al., “ROS: An Open-Source Robot Operating System”, *ICRA Workshop on open source software*, Vol. 3, No. 3.2, p. 5., 2009

[5] B. Benjamin, D. Bernhard, S. Peter, "Secure communication for the robot operating system" *Robotics and Autonomous Systems*, Vol. 98, pp. 192-203., 2017

[6] E. Ciliendo, T. Kunimasa, *Linux Performance and Tuning Guidelines*, pp. 41, IBM Redbooks, 2007