Future Generation Computer Systems 56 (2016) 171–178

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

RT-ROS: A real-time ROS architecture on multi-core processors

Hongxing Wei[a,1], Zhenzhou Shao[b], Zhen Huang[a], Renhai Chen[d], Yong Guan[b], Jindong Tan[c,1], Zili Shao[d,*,1]

[a] School of Mechanical Engineering and Automation, Beihang University, Beijing, 100191, PR China
[b] College of Information Engineering, Capital Normal University, Beijing, 100048, PR China
[c] Department of Mechanical, Aerospace, and Biomedical Engineering, The University of Tennessee, Knoxville, TN, 37996-2110, USA
[d] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China

ARTICLE INFO

Article history:
Received 6 February 2015
Received in revised form
20 April 2015
Accepted 12 May 2015
Available online 9 June 2015

Keywords:
Real-time operating systems
Robot Operating Systems
Multi-core processors

ABSTRACT

ROS, an open-source robot operating system, is widely used and rapidly developed in the robotics community. However, running on Linux, ROS does not provide real-time guarantees, while real-time tasks are required in many robot applications such as robot motion control. This paper for the first time presents a real-time ROS architecture called RT-RTOS on multi-core processors. RT-ROS provides an integrated real-time/non-real-time task execution environment so real-time and non-real-time ROS nodes can be separately run on a real-time OS and Linux, respectively, with different processor cores. In such a way, real-time tasks can be supported by real-time ROS nodes on a real-time OS, while non-real-time ROS nodes on Linux can provide other functions of ROS. Furthermore, high performance is achieved by executing real-time ROS nodes and non-real-time ROS nodes on different processor cores. We have implemented RT-ROS on a dual-core processor and conducted various experiments with real robot applications. The experimental results show that RT-ROS can effectively provide real-time support for the ROS platform with high performance by exploring the multi-core architecture.

1. Introduction

ROS, an open-source robot operating system, has been being rapidly developed and widely used in the robotics community [1]. Based on the ROS framework, many researchers have developed their software for diverse robots such as Barrett WAM [2] and Raven-II [3]. However, ROS runs on Linux, and cannot provide real-time guarantees. This limits its usage, as real-time tasks are required in many robot applications such as robot motion control. Therefore, it becomes a key issue to make ROS be real-time. On the other hand, multi-core processors offer a promising platform for robot applications. Compared to the traditional robot computing platform with separated host and guest systems, a multi-core processor can provide more powerful computing capacity and less communication overhead. Thus, it is also vitally important to effectively run ROS on multi-core processors by exploring the multi-core architecture for robot applications. This paper focuses on solving the problem of making ROS be real-time with high performance on multi-core processors.

To make ROS be real-time, a common approach is to run real-time tasks on guest embedded systems and run non-real-time tasks on a host system such as in ROS Industrial and ROS *Bridge* [4]. However, by separating ROS tasks into different computing systems, it not only introduces big communication overhead but also increases the manufacturing cost. In fact, a multi-core processor such as Intel Pentium multi-core processors is powerful enough to run both real-time and non-real-time tasks of ROS. Implementing ROS based on a system with a multi-core processor can help reduce communication overhead by replacing inter-system communication with inter-core communication, decrease the system cost and simplify the system design. However, it is still an open issue for how to make ROS be real-time on multi-core processors.

There are challenges to make ROS be real-time on multi-core processors. First, considering the portability, a real-time OS environment should be provided to support the execution of real-time ROS tasks. It is challenging to run both a real-time OS and a general-purpose OS without interfering each other on multi-core processors. In particular, it is not trivial to provide a mechanism so interrupts, devices and other hardware resources can be separated

* Corresponding author.
E-mail addresses: weihongxing@buaa.edu.cn (H. Wei), guanyxxxy@263.net (Y. Guan), tan@utk.edu (J. Tan), cszlshao@comp.polyu.edu.hk (Z. Shao).
1 Member, IEEE.

http://dx.doi.org/10.1016/j.future.2015.05.008
0167-739X/© 2015 Elsevier B.V. All rights reserved.

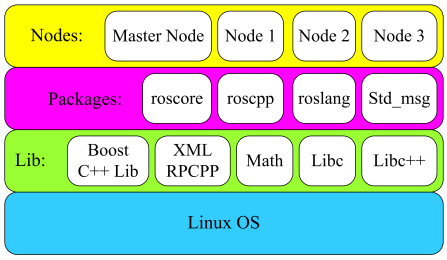

Fig. 1. The software architecture of ROS.

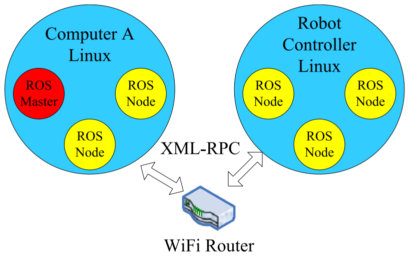

Fig. 2. The remote control model of ROS.

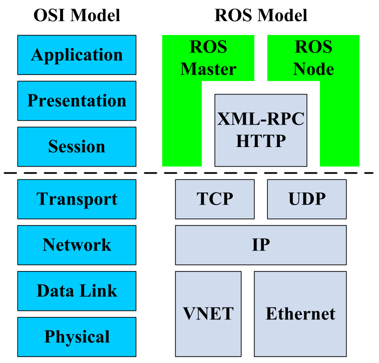

Fig. 3. The communication model of ROS and its corresponding OSI model.

and isolated to support real-time and non-real-time ROS tasks. Second, we also need to provide effective and efficient communication mechanisms between real-time and non-real-time tasks.

This paper for the first time presents a real-time ROS architecture called RT-ROS on multi-core processors. RT-ROS provides an integrated real-time/non-real-time task execution environment so real-time and non-real-time ROS nodes can be separately run on a real-time OS and Linux, respectively, with different processor cores. In such a way, real-time tasks can be supported by real-time ROS nodes on a real-time OS, while non-real-time ROS nodes on Linux can provide other functions of ROS. In RT-ROS, we develop a hybrid OS platform that can support the execution of real-time and non-real-time OSes. RT-ROS provides a mechanism in its hybrid OS so processor cores and other hardware resources such as interrupts and devices are divided and isolated, by which real-time ROS tasks on a real-time OS and non-real-time ROS tasks on a general-purpose OS can be run separately without interfering each other. Furthermore, in RT-ROS, we develop an efficient communication mechanism between real-time and non-real-time OSes.

We have implemented RT-ROS on an Intel dual-core processor. Nuttx [5], an open source real-time OS, is chosen as the real-time platform, and Linux is used as the general-purpose OS. We further implement the RT-ROS system in the industrial controller of a 6-DOF modular manipulator. The experimental results show that RT-ROS can effectively provide real-time support for the ROS platform with high performance by exploring the multi-core architecture.

The remainder of this paper is organized as follows. Section 2 introduces the background. Sections 3 and 4 present the design and implementation of RT-ROS, respectively. Evaluation is presented is Section 5. Section 6 introduces the related work. Finally, the conclusions are drawn in Section 7.

2. Background

In this section, we provide the background. We first introduce ROS and then Nuttx that is used as the real-time OS in the implementation of RT-ROS.

2.1. ROS

ROS (Robot Operating System) is an open-source and reusable software platform providing libraries, tools and conventions that can help to create high-performance robot applications quickly and easily. It provides standardized interfaces for hardware control, tools for creating, debugging, distributing and running procedures, and libraries for developing programs. So far, about 500 packages have been made available in ROS from approximately 30 institutions [6–8].

ROS is a modular software platform installed on the Linux operating system. It contains a number of software modules encapsulated as nodes, including the master node and the functional nodes. Fig. 1 gives the software architecture of ROS, which consists of libraries, packages and nodes installed on Linux. Such a modular architecture supports distributed network communication. Like the router in a LAN, the master node administrates and monitors the running of the functional nodes and their peer-to-peer communications.

Fig. 2 shows a typical remote control model of ROS. The ROS nodes in a robot are connected to these in the remote computer through the WIFI router. The communications among nodes are realized by XML/RPC (Remote Procedure Call) based on the TCP/IP protocol.

Fig. 3 illustrates the ROS communication model and its corresponding OSI model. It can be seen that an application layer based on the XML-RPC HTTP protocol is constructed on the TCP/IP architecture. Therefore, messages transmitted among nodes are not data packages but webpage files based on the http protocol.

2.2. Nuttx

In order to construct the hybrid operating system, we need a portable real-time operating system architecture. There are many real-time operating systems such as VxWorks, QNX, eCOS, and μcOS-II. Considering the portability of ROS, we choose Nuttx as our real-time OS platform.

Nuttx is an open-source embedded real time operating system (RTOS) developed by Gregory Nutt [5]. Nuttx supports the Posix and ANSI standards, and can be applied in microcontrollers from 8-bit to 32-bit. In addition, by adopting the standard APIs from Unix and other common RTOSes such as VxWorks, Nuttx also provides some functionalities not available under the Posix and ANSI standards. Compared with other real-time operating systems such as VxWorks, Windows CE and μC/OS-II, Nuttx has the following advantages:

(1) Various system services. Nuttx provides a number of system supports such as the UIP protocol stack, networking, device drivers and virtue file system, which are useful for application development.

(2) Small footprint. Nuttx only requires very little memory. For example, 4 MB memory is enough to run Nuttx.

(3) Easy extension. It is convenient to extend Nuttx to new processor architectures such as SoC architectures. This makes it possible to port real-time ROS nodes among different multi-core processors.