A Profiling Framework in Linux/RK and its Application

Hyoseung Kim, Junsung Kim, Ragunathan (Raj) Rajkumar
Carnegie Mellon University, Pittsburgh, PA 15213
{hyoseunk, junsungk, raj}@ece.cmu.edu

ABSTRACT

We developed a profiling framework using Linux/RK, and we integrated it into a self-driving car software system as a proof-of-concept implementation. The demonstration shows that our two-level profiling interface on Linux/RK can be used to monitor and analyze the runtime resource usage of a complex system that runs a multitude of algorithms including sensor fusion, computer vision and path planning.

1. INTRODUCTION

Advances in complex embedded real-time systems pose challenges in monitoring and analyzing runtime resource usage. Understanding the runtime behavior of a system is important for the design, analysis and implementation of a real-time system. To address this issue, we developed a profiling framework using Linux/RK [3]. We also demonstrate a proof-of-concept implementation on a real-world embedded real-time system [5].

2. LINUX/RK PROFILING FRAMEWORK

Linux/RK [3] provides guaranteed and enforced access to system resources for real-time tasks by using the resource reservation approach [4]. The key resource management primitives of Linux/RK are *reserve* and *resource set*. A reserve is a well-defined representation of a system resource such as CPU, memory and network. In case of CPU, a CPU reserve is specified with (ϕ, C, T, D), where ϕ is a release offset, C is the worst-case execution time, T is a period, and D is a relative deadline. A resource set is a container of multiple reserves. Zero, one or more tasks can be attached to a single resource set, which provides the exclusive use of the reserved resources in the resource set.

While Linux/RK had its own resource usage accounting capability, it was not sufficient to precisely understand the runtime resource usage of a real-time system. Hence, we implemented a profiling framework on Linux/RK. The framework provides a two-level profiling interface that supports *reserve-level* and *task-level* profiling. Figure 1 shows an example of the two-level profiling for CPU resources. At the reserve level, the profiler generates the release time, the resource usage and the completion time of each instance of a reserve. At the task level, the profiler generates the execution information of each task instance. If a reserve is enforced, i.e., a reserve uses up its given resources, the enforcement information is delivered to reserve's tasks via an OS signal and recorded by the profiler. Double-buffering is used to minimize the temporal interference from a user-level reader task, and `tsc` and `hrtimer` are used for time measurement. In our microbenchmark, the computational overhead for recording profile data was negligible, 194ns in average on the Intel i7 2.4 GHz processor.

Application: By using the profiled data, an application developer can easily analyze the resource usage and (re-)configure the amount of resources allocated to tasks. For instance, the occurrence of the enforcement event indicates

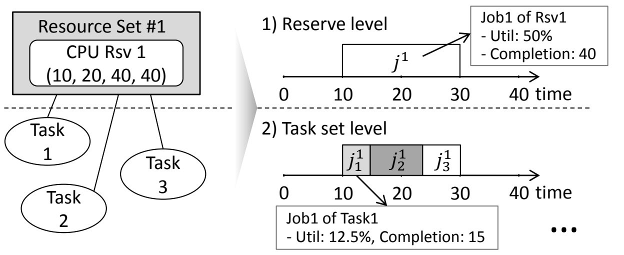

Figure 1: An example of our profiling framework

the lack of resources, so more resources may need to be assigned. The over-assigned resources can be identified by the resource usage of either the reserve level or the task level. We expect that our profiling mechanism can be effectively used as a cost-effective way to estimate the resource usage of a complex real-time system and to verify the timeliness of the system. A dynamic and automatic resource reconfiguration scheme based on the profiling mechanism is a part of our future work.

3. DEMONSTRATION

The Linux/RK profiling framework is integrated into an autonomous vehicle software system developed at CMU [2, 1]. Three new features have been added to the system to profile tasks running on a self-driving car [5]. The autonomous driving software is composed of many tasks on distributed machines. Each task is periodically executed through the task library enforced by Linux/RK and managed by a daemon running on each machine. The status of each task is monitored by user control software. The following modifications are made to the system:

- The task library is modified to enable/disable task profiling according to the configuration file.
- If task profiling is enabled for a task, the task periodically reports its profiling data such as a pair of release time and computation time to a task collecting all profiling data from different machines.
- The collected data can be stored into a file and displayed through a widget showing task information.

We demonstrate a scenario, where a self-driving car travels around a test loop in Hazelwood, Pittsburgh, PA. During the demo, the user interface shows the utilization of each task as a graph while displaying its maximum and minimum values.

4. REFERENCES

[1] J. Kim, G. Bhatia, R. R. Rajkumar, and M. Jochim. SAFER: System-level Architecture for Failure Evasion in Real-time Applications. In *the 33rd IEEE RTSS*, 2012.
[2] M. McNaughton et al. Software infrastructure for an autonomous ground vehicle. *Journal of Aerospace Computing, Information, and Communication*, 5(1):491 – 505, December 2008.
[3] S. Oikawa and R. Rajkumar. Linux/RK: A Portable Resource Kernel in Linux. In *IEEE RTSS Work-In-Progress*, 1998.
[4] R. Rajkumar et al. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *SPIE/ACM Conference on Multimedia Computing and Networking*, 1998.
[5] C. Urmson, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(1):425–466, June 2008.