Analyzable and Practical Real-Time Gang Scheduling on Multicore Using *RT-Gang*

Waqar Ali, Michael Bechtel, Heechul Yun
University of Kansas
{wali, mbechtel, heechul.yun}@ku.edu

In this *presentation*, we will introduce a new real-time gang scheduling framework in Linux, called RT-Gang [2], and provide a brief tutorial and a demo of using the framework in a self-driving car application [3].

Emerging safety-critical real-time control systems in automotive and aviation applications often consist of a number of highly computationally expensive and data intensive workloads (e.g., deep neural networks) with strict real-time requirements for agile control (e.g., 50Hz). Guaranteeing timely execution of these real-time tasks is an important requirement for safety of the system. However, it is challenging to provide such a guarantee on today's highly integrated embedded computing platforms because they often show unpredictable and extremely poor worst-case timing behaviors that are hard to understand, analyze, and control [4], [8]—chiefly due to interference in shared memory hierarchies. Broadly, timing unpredictability is a serious problem especially in automotive and aviation industries. For example, Bosch, a major automotive supplier, reported "predictability on high-performance platforms" as a major industrial challenge for which the industry is actively seeking solutions from the research community [6]. In aviation, the problem was dubbed as "one-out-of-m" problem [7] because the current best practice for certification, which requires *evidence of bounded interference*, is to disable all but one core of a multicore processor [5].

RT-Gang [2] is a new real-time gang scheduling framework implemented in Linux to address the timing unpredictability problem on COTS multicore platforms for safety-critical real-time applications. In RT-Gang, all threads of a parallel real-time task form a real-time gang and the scheduler globally enforces a *one-gang-at-a-time* scheduling policy. When a real-time task is released, all of its threads are scheduled simultaneously if it is the highest priority real-time task, or none at all if a higher priority real-time task is currently in execution. Any idle cores, if exist, can be used to schedule best-effort tasks but their shared memory access rates are strictly regulated by a memory throttling mechanism to bound their impact to the real-time task. Specifically, each real-time task defines its tolerable maximum memory bandwidth budget, which is strictly enforced by a kernel level regulator for any co-scheduled best-effort tasks. (see Figure 1.)

RT-Gang eliminates the problem of contention in the shared memory hierarchy between real-time tasks by executing only one real-time task at any given time, which effectively transforms parallel real-time task scheduling on a multicore into the well-understood uni-core real-time scheduling problem. Because of the strong temporal isolation guarantee offered by RT-Gang, a real-time task's worst-case execution time (WCET) can be tightly bounded without making strong assumptions about the underlying hardware. Thus, RT-Gang can improve system schedulability while providing a mechanism to safely utilize all cores of a multicore platform.

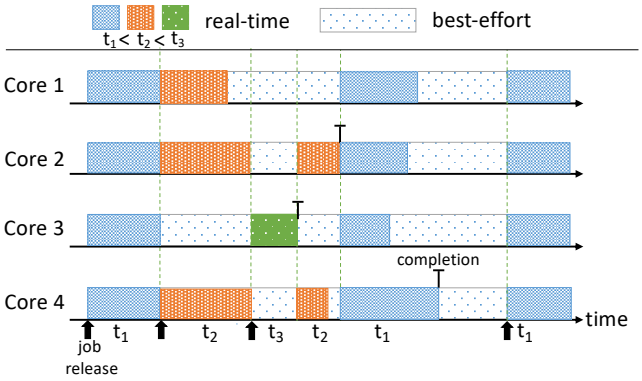

Fig. 1. Illustration of RT-Gang

RT-Gang is currently implemented as a "feature" of the standard Linux SCHED_FIFO real-time scheduler (kernel/sched/rt.c), which can be enabled or disabled dynamically at run-time [1]. In this presentation, we will provide a quick tutorial on how to use the feature, and demonstrate its effects on a real self-driving car application [3], which uses deep neural networks (processed by TensorFlow).

References

[1] RT-Gang code repository. https://github.com/CSL-KU/RT-Gang.
[2] W. Ali and H. Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.
[3] M. G. Bechtel, E. McEllhiney, and H. Yun. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.
[4] M. G. Bechtel and H. Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.
[5] Certification Authorities Software Team. CAST-32A: Multi-core Processors. Technical report, Federal Aviation Administration (FAA), 2016.
[6] A. Hamann. Industrial challenges: Moving from classical to high performance real-time systems. In *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2018.
[7] N. Kim, B. C. Ward, M. Chisholm, J. H. Anderson, and F. D. Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. *Real-Time Systems*, 53(5):709–759, 2017.
[8] P. K. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.