Augmenting Criticality-Monotonic Scheduling with Dynamic Processor Affinities

Björn B. Brandenburg
bbb@mpi-sws.org

Keywords: multiprocessors, arbitrary processor affinities (APAs)

Extended Abstract

Consider the problem of scheduling a dual-criticality workload consisting of high- and low-criticality sporadic real-time tasks on top of a fixed-priority (FP) scheduler. Each high-criticality (HC) task $T_i$ has both a high- and a low-criticality WCET estimate, denoted $e^L_i$ and $e^H_i$, resp., and low-criticality (LC) tasks are required to meet their deadlines only if no HC task exceeds its LC WCET estimate.

From a pragmatic point of view, FP scheduling with criticality-monotonic priorities [1], where HC tasks have higher priority than LC tasks, holds considerable appeal: it is simple, provides obvious isolation for HC tasks, and imposes no runtime overheads. Unfortunately, as LC tasks may be more urgent than HC tasks (i.e., they may have shorter periods or more constraining deadlines), it is not always feasible to assign criticality-monotonic priorities [1]. For example, the task set $\tau_1 = \{T_a, T_b\}$ (as specified in Fig. 1), which consists of a LC task $T_a$ that is urgent (i.e., it has a short period $p_a = 2$) and a HC task $T_b$ that is less urgent ($p_b = 10$) but more costly ($e^L_b = 3$), cannot be scheduled on a uniprocessor with criticality-monotonic priorities: the LC task $T_a$, if given a lower priority than $T_b$, may miss deadlines even if no job of $T_b$ exceeds $e^L_b$.

Similar urgency vs. criticality conflicts also arise on multiprocessors. For instance, the task set $\tau_2 = \{T_a, T_b, T_c, T_d\}$ cannot be scheduled with criticality-monotonic priorities on $m = 2$ cores using either global or partitioned FP scheduling: under global scheduling, the HC tasks $T_b$ and $T_d$ can cause the more-urgent LC tasks $T_a$ and $T_c$ to miss deadlines even with LC execution costs, and under partitioned scheduling, $T_b$ and $T_d$ need to be assigned to different partitions, but neither can be co-located with $T_a$ or $T_c$. However, while scheduling $\tau_1$ with criticality-monotonic priorities is infeasible on a uniprocessor, $\tau_2$ can be scheduled with criticality-monotonic priorities on two processors—provided processor affinities are used to shield urgent tasks in the LC case.

Exploiting Arbitrary Processor Affinities (APAs). Contemporary OSs such as Linux, Windows, QNX, or VxWorks provide flexible APIs to explicitly set a task’s processor affinity, which is the set of processors on which it may execute. In particular, task affini-
ties can be restricted to arbitrary processor sets and changed at arbitrary times during runtime. This can be exploited to render criticality-monotonic scheduling feasible.

Consider the following strategy for scheduling $\tau_2$ on two processors $P_1$ and $P_2$: 

1. Tasks are assigned criticality-monotonic priorities.
2. $T_a$ and $T_c$ may execute on both $P_1$ and $P_2$.
3. $T_b$ and $T_d$ may initially execute only on processor $P_1$.
4. When a HC job $J_x$ of $T_b$ (resp., $T_d$) fails to complete after $e_{L}^x$ (resp., $e_{L}^d$) time units, it updates its processor affinity to include both $P_1$ and $P_2$. (The processor affinity of any other task is not changed.)
5. A HC task’s affinity is reset when it completes its job.

A possible schedule is shown in Fig. 1: at time 3, when it becomes known that $T_b$’s job requires more than $e_{L}^b = 3$ time units to complete, it relaxes its processor affinity to include $P_1$ and $P_2$. Consequently, under a FP scheduler with strong APA semantics [2] — which, intuitively, is an APA scheduler that shifts higher-priority tasks from one processor to another if that is required to enable lower-priority tasks with more-constraining affinities to be scheduled — $T_b$ shifts to $P_2$, which enables $T_d$ to be scheduled on $P_1$. As $T_b$ handles its increased demand on $P_2$, $T_d$ is protected from undue interference. LC tasks are not dropped, but may temporarily incur deadline misses.

Remarks and outlook. We have observed that an APA interface — readily available in current, already certified RTOSs — allows the timeliness requirements of urgent LC tasks to be reconciled with the desirable simplicity of criticality-monotonic scheduling. The sketched approach offers several practical benefits: HC tasks exceeding their LC WCET are effectively given a “dedicated” processor to cope with increased demand; only the currently executing task’s affinity is adapted, which keeps runtime overheads low and independent of the number of tasks; there is no “mode change” and LC tasks are not abandoned, just temporarily delayed; and budget enforcement is not required.

Of course, the above example works only because of simplifying assumptions. We believe, however, that it is possible to generalize the approach to an arbitrary number of HC tasks and also to weak APA schedulers [2] such as those found in QNX and Linux.

References
