Work-in-Progress: Strong APA Scheduling in a Real-Time Operating System

Richi Dubey
Birla Institute of Technology and Science Pilani
Goa, India
f20170099@goa.bits-pilani.ac.in

Vijay Banerjee, Sena Hounsinou, Gedare Bloom
University of Colorado Colorado Springs
Colorado Springs, CO, USA
(vbanerje,shoueto,gbloom)@uccs.edu

ABSTRACT

Arbitrary processor affinities are used in multiprocessor systems to specify the processors on which a task can be scheduled. However, affinity constraints can prevent some high priority real-time tasks from being scheduled, while lower priority tasks execute. This paper presents an implementation and evaluation of the Strong Arbitrary Processor Affinity scheduling on a real-time operating system, an approach that not only respects user-defined affinities, but also supports migration of a higher priority task to allow execution of a task limited by affinity constraints. Results show an improvement in response and turnaround times of higher priority tasks.

CCS CONCEPTS

• **Computer systems organization → Real-time operating systems**.

KEYWORDS

RTOS, RTEMS, SMP, APA Scheduling

1 INTRODUCTION

In symmetric multiprocessing (SMP) systems, each processor has direct access to system resources and is treated as an independent unit by the operating system (OS). Many OSs using SMP also allow setting a *processor affinity* for a task, i.e., the set of processors that can execute it. Affinity scheduling reduces task migrations and can result in better performance. However, it can also affect the system's schedulability: a task can miss its deadline if all processors in its affinity set are executing higher priority tasks, even if there is an idle processor in the affinity set of a higher priority task.

In this work, we present the implementation of an arbitrary processor affinity (APA) scheduler on the Real-Time Executive for Multiprocessor Systems (RTEMS) that allows shifting of tasks without violating original affinity restrictions, thus improving the response-time and turnaround times for higher priority tasks. RTEMS is an SMP-supported POSIX-compliant real-time OS that supports task affinity through its SMP framework from the application layer using *rtems_task_set_affinity()* [1]. In contrast, the Priority Affinity Scheduler in RTEMS guarantees a task is scheduled on a processor in its affinity set, but does not migrate higher priority tasks executing on a processor in the affinity set of a newly arrived task.

EMSOFT'21 Companion, October 8–15, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8712-5/21/10…$15.00
https://doi.org/10.1145/3477244.3477623

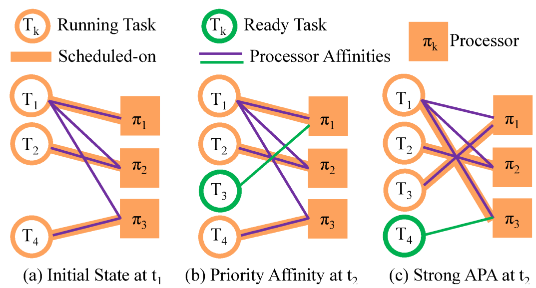

Figure 1: Priority Affinity vs. Strong APA Scheduling: At time t_1 (a), T_1, T_2 and T_4 are assigned to π_1, π_2, and π_3 respectively. (b) Weak APA: T_3 is blocked by T_1 executing on π_1. (c) Strong APA: T_1 migrates to π_3, blocks T_4 and T_3 gets π_1

Affinity scheduling has multiple use cases such as providing security against cache side-channel attacks and maintaining cache locality.Migration allows an arriving task to displace a running task from its current processor to another processor in its affinity. In *Weak APA*, a task cannot be scheduled if all processors in its affinity are executing higher-priority tasks. *Strong APA* allows the scheduling of a lower priority task with affinity constraints by dislodging higher-priority tasks to other processors in their affinity set [2]. In this work, we consider a system with a set of n real-time tasks that run on a set of m identical processors $\Pi = \{\pi_1, ..., \pi_m\}$. Each task T_i ($1 \leq i \leq n$) has a user-defined affinity $\alpha_i \subseteq \Pi$, a scheduler node N_i, and a priority $prio_i$. Fig. 1 illustrates the difference between Weak APA and Strong APA scheduling with $n = 4$ and $m = 3$. Priorities are in decreasing order (i.e., T_1 has highest priority). α_1, α_2, α_3 and α_4 are $\{\pi_1, \pi_2, \pi_3\}$, $\{\pi_2\}$, $\{\pi_1\}$, and $\{\pi_3\}$, respectively. T_1, T_2, T_4 are released at time t_1, and T_3 at time t_2 ($t_2 > t_1$).

2 STRONG APA SCHEDULER ON RTEMS

At a task arrival or departure, the Strong APA scheduler uses task reachability to update the scheduled task set.

Definition 2.1. A *minimum reachable unit* is a triplet $U_i^j = < T_i, \pi_k, T_j >$ consisting of a *source* task T_i, a *destination* task T_j, and a processor π_k such that $\pi_k \in \alpha_i \cap \alpha_j$. A task T_j is called a *reachable* task from T_i, if there exists an ordered reachability set $R_{i,j} = \left\{U_i^a, U_a^b, ..., U_c^d, U_d^j\right\}$, such that the ordered set $|R_{i,j}| > 0$ and $\{a, b, \ldots, c, d\} \in [1, n]$.

Task Arrival: When a task T_i arrives, the scheduler finds $R_{i,lo}$, such that $\forall U_a^b \in R_{i,lo}$, T_b is executing on processor $\pi_k \in U_b^a$, and T_{lo} is the lowest priority scheduled reachable task. The handling of task

Algorithm 1 _Scheduler_strong_APA_Enqueue

Input: T_i **Output:** $stat$

1: $cpu\_to\_preempt \leftarrow$ nil, $front \leftarrow 0$, $rear \leftarrow -1$
2: $prio_{lo} \leftarrow HIGHEST\_PRIORITY\_IN\_SYSTEM$
3: **for** $\pi_i \in \alpha_i$ **do**
4: queue[++rear] $\leftarrow \pi_i$, mark π_i as visited
5: preempting_node(π_i) $\leftarrow N_i$
6: **while** $front \leq rear$ **do**
7: $\pi_{cur} \leftarrow queue[front++]$, $T_{cur} \leftarrow$ Task running on π_{cur}
8: **if** $prio_{cur} < prio_{lo}$ **then**
9: $prio_{lo} \leftarrow prio_{cur}$, $cpu\_to\_preempt \leftarrow \pi_{cur}$
10: **for** $\pi_t \in \alpha_{cur}$ **do**
11: **if** $T_{cur} \neq$ Idle_task **and** π_t not visited **then**
12: $queue[++rear] \leftarrow \pi_t$, mark π_t as visited
13: $preempting\_node(\pi_t) \leftarrow N_{cur}$
14: **if** ($prio_{lo} > prio_i$) **then** $stat \leftarrow$ Add N_i to Ready Queue
15: **else**
16: $N_{preempt} \leftarrow preempting\_node(cpu\_to\_preempt)$
17: **while** $N_{preempt} \neq N_i$ **do**
18: $next\_cpu\_to\_preempt \leftarrow$ cpu running $N_{preempt}$
19: $Preempt(cpu\_to\_preempt, N_{preempt})$
20: $cpu\_to\_preempt \leftarrow next\_cpu\_to\_preempt$
21: $N_{preempt} \leftarrow preempting\_node(cpu\_to\_preempt)$
22: $stat \leftarrow$ Add N_i to Scheduled Queue

arrival is shown in Alg. 1. If $prio_{lo} < prio_i$, the scheduler preempts T_{lo} and allocates the processor from the first element of $R_{i,lo}$ to T_i by shifting the tasks along the path based on $R_{i,lo}$. On T_i's arrival, the scheduler calls *_Scheduler_strong_APA_Enqueue*() which maintains a queue of processors initialized with α_i (lines 2 to 5). To find T_{lo}, for each processor π_{cur} in the queue, if the task T_{cur} running on π_{cur} is the lowest priority task seen so far, then π_{cur} is marked for preemption. Further, if T_{cur} is not the idle task, then each processor $\pi_t \in \alpha_{cur}$ is inserted to the queue and N_{cur} marked as the preempting node for π_t (lines 6 to 13). Once T_{lo} is identified, if its priority is higher than T_i's, then T_i is enqueued to the ready queue (line 13). Otherwise T_i is assigned a processor by backtracking from T_{lo}'s processor: *cpu_to_preempt* (lines 15 to 21).

Task Departure: When task (T_{depart}) departs processor (π_{depart}), Alg. 2 finds $R_{depart,hi}$, such that $\forall U_a^b \in R_{depart,hi}$, T_a is executing on processor $\pi_k \in U_b^a$, and T_{hi} is the highest priority ready reachable task. The scheduler schedules T_{hi} and allocates a task to π_{depart} by shifting the tasks along the path based on $R_{depart,hi}$.

A queue of processors is maintained and initialized with π_{depart}. For each processor π_{frt} in the queue, the function identifies any task T_{cur} in the system that has π_{frt} in its affinity. If T_{cur} is scheduled, then the processor that it is executing on, π_{cur}, is inserted into the queue. Else, if T_{cur} is the highest priority ready task witnessed so far, it is marked as (T_{hi}). In both cases, π_{frt} is marked as the cpu that N_{cur} would preempt (lines 3 to 10). Once T_{hi} is identified, it is allocated a processor by backtracking through the path from T_{hi} to π_{depart}, allocating a task to π_{depart} as well (lines 12 to 17).

Evaluation and Preliminary Results. We evaluated the Strong APA implementation on QEMU with ARM target *realview-pbx-a9*.

Algorithm 2 _Scheduler_strong_APA_Schedule_highest_ready

Input: T_{depart}, π_{depart}

1: front $\leftarrow 0$, rear $\leftarrow 0$, $queue[rear] \leftarrow \pi_{depart}$
2: $prio_{hi} \leftarrow LOWEST\_PRIORITY\_IN\_SYSTEM$
3: **while** front $\leq$ rear **do**
4: $\pi_{frt} \leftarrow queue[front++]$
5: **for** each task T_{cur} such that $\pi_{frt} \in \alpha_{cur}$ **do**
6: **if** T_{cur} is scheduled **then**
7: $\pi_{cur} \leftarrow$ Processor T_{cur} is executing on
8: queue[++rear] $\leftarrow \pi_{cur}$, $N_{cur}.to\_preempt \leftarrow \pi_{frt}$
9: **else if** $prio_{cur} > prio_{hi}$ **then** $prio_{hi} = prio_{cur}$
10: $T_{hi} \leftarrow T_{cur}$; $N_{cur}.to\_preempt \leftarrow \pi_{frt}$
11: $cur\_node \leftarrow N_{hi}$, $cur\_cpu \leftarrow N_{cur\_node}.to\_preempt$
12: **while** $cur\_cpu \neq \pi_{depart}$ **do**
13: $next\_node \leftarrow$ Node for task scheduled on cur_cpu
14: $Preempt(cur\_cpu, cur\_node)$
15: $cur\_node \leftarrow next\_node$
16: $cur\_cpu \leftarrow N_{cur\_node}.to\_preempt$
17: $Preempt(cur\_cpu, cur\_node)$

	Avg. Response time(in μs)		Avg. Turnaround time(in μs)	
Task	PA	SAPA	PA	SAPA
1	141.05	306.95	3086783.75	3230183.83
2	207.95	614.78	3197499.71	3136835.88
3	3088997.91	198.45	6373255.57	3169505.46
4	134.72	325.54	3175646.44	6160782.63

Table 1: Response Time and Turnaround Time

We compared the average response time and average turnaround time of the tasks under Priority Affinity (PA) and Strong APA (SAPA) scheduling. Table 1 shows average times over 100 runs. We note a stark difference in the response time of T_3 due to the fact that T_3 was blocked by T_1 in PA scheduling, while SAPA allowed T_1 to migrate to π_3, allowing T_3 to execute. Similarly, the average turnaround time demonstrates that T_3 is scheduled when it arrives with SAPA scheduling and T_4, which is the lower priority task, was blocked until T_1 finished its execution.

3 CONCLUSION AND FUTURE WORK

We presented an implementation of Strong APA scheduling on the RTEMS real-time OS. The evaluation results show that Strong APA scheduling has lower response and turnaround times for higher priority tasks. As a next step, we intend to evaluate context switch overhead due to task migration and the performance of the scheduler over a large task set.

ACKNOWLEDGMENTS

This work is supported by NSF OAC-2001789 and Colorado State Bill 18-086. We thank Sebastian Huber and the RTEMS community.

REFERENCES

[1] Gedare Bloom, Joel Sherrill, Tingting Hu, and Ivan Cibrario Bertolotti. 2020. *Real-Time Systems Development with RTEMS and Multicore Processors.* CRC Press. https://doi.org/10.1201/9781351255790

[2] Felipe Cerqueira, Arpan Gujarati, and Björn B. Brandenburg. 2014. Linux's Processor Affinity API, Refined: Shifting Real-Time Tasks Towards Higher Schedulability. In *2014 IEEE Real-Time Systems Symposium.* 249–259. https://doi.org/10.1109/RTSS.2014.29