Linux’s Processor Affinity API, Refined: *Shifting Real-Time Tasks towards Higher Schedulability*

Felipe Cerqueira, Arpan Gujarati, Björn Brandenburg
Task Migration under Current RTOSs: Arbitrary Processor Affinities (APA)

Standard API provided by Linux, QNX, VxWorks, …
Use Cases of Processor Affinities

Security

Isolate tasks to prevent cache side-channel attacks

Cache Locality

Avoid migration-related cache misses

Energy Efficiency

Restrict non-critical tasks to small, power-efficient cores

and more…
Use Cases of Processor Affinities

- Security
- Cache Locality
- Energy Efficiency
- and more…

Application-specific affinity requirements may render the system unschedulable.
Affinities can cause Deadline Miss

Linux
Affinities can cause Deadline Miss

Tasks are released

Linux
Affinities can cause Deadline Miss

Processor idles, but Task 3 cannot execute there
Affinities can cause Deadline Miss

Task 3 misses deadline!
Can we improve the ability to meet deadlines without violating the affinity assignment?
Shifting Tasks to Improve the Schedule

Our Approach

Affinities

Linux

Our Approach
Our Approach

Task 1 shifts to the other processor so that Task 3 can execute
Shifting Tasks to Improve the Schedule

Our Approach

No deadline misses for Task 3!
New Migration Semantics for APA Scheduling via Task Shifting

Processors

restricted to this processor
New Migration Semantics for APA Scheduling via Task Shifting

Shifting Migration

Processors

T_3

T_1
New Migration Semantics for APA Scheduling via Task Shifting

Shifting migrations free processors for a restricted task

→ Improved Schedulability
Full API Compatibility

- Affinity API
- Scheduler

API compatibility
- No affinity violations
- Improved schedulability

Only change: when tasks migrate

API unmodified!
Similar Problem in Operations Research

Assignment Problem with Seniority Constraints [Caron et al 1999]

Problem: Assign jobs in a hospital

Constraints:
(1) Jobs require qualification
(2) Senior employees have preference

Two variants

Weak Seniority

Strong Seniority

Contributions of our Paper

1) Distinction between:

APA scheduling without shifting \iff Weak APA
APA scheduling with shifting \iff Strong APA
Contributions of our Paper

1) Distinction between:
   - APA scheduling without shifting ⇔ Weak APA
   - APA scheduling with shifting ⇔ Strong APA

2) Formalization of strong APA scheduling based on Bipartite Matching
Contributions of our Paper

1) Distinction between:
   - APA scheduling without shifting \iff Weak APA
   - APA scheduling with shifting \iff Strong APA

2) Formalization of strong APA scheduling based on Bipartite Matching

3) Dynamic algorithm for task shifting
Contributions of our Paper

1) Distinction between:
   APA scheduling without shifting \iff Weak APA
   APA scheduling with shifting \iff Strong APA

2) Formalization of strong APA scheduling based on Bipartite Matching

3) Dynamic algorithm for task shifting

4) Schedulability Analysis for strong APA Scheduling
This Talk

Limitations of current APA schedulers

How to perform task shifting

Schedulability Analysis

Evaluation
This Talk

Limitations of current APA schedulers

How to perform task shifting

Schedulability Analysis

Evaluation
Limitations of Current APA Schedulers

Example where Linux will violate task priorities
Task $T_3$ arrives

Linux locally checks if there is a CPU to be preempted in $T_3$’s affinity.
Linux does not Schedule the Task!

Linux locally checks if there is a CPU to be preempted in $T_3$’s affinity.

No preemption! CPU 1 already has a higher-priority task.
But there is a Better Schedule

(Task priorities: $T_1 < T_2 < T_3 < T_4$)
Global Decision is Required to Compute the Correct Schedule

Task priorities must be respected

Processor utilization must be maximized

Linux does not always guarantee both!
This Talk

- Limitations of current APA schedulers
- How to perform task shifting
- Schedulability Analysis
- Evaluation
Scheduling as a Bipartite Matching

Any matching in the graph is a valid scheduler state.
Maximum Bipartite Matching?

A maximum bipartite matching maximizes processor utilization
Maximum Bipartite Matching?

A maximum bipartite matching maximizes processor utilization …but does not enforce task priorities.
Maximum Vertex-Weighted Bipartite Matching (MVM)

If we map task priorities to vertex weights, **MVM** is the optimal scheduling decision.
If we map task priorities to vertex weights, MVM is the optimal scheduling decision.

Maximum Vertex-Weighted Bipartite Matching (MVM)

Scheduling decisions for strong APA can be computed with existing graph algorithms.
Scheduling Decisions must be Fast!

- Scheduler is a critical part of an OS
- Computing an MVM from scratch is costly
Scheduling Decisions must be Fast!

• Scheduler is a critical part of an OS
• Computing an MVM \textit{from scratch} is costly

Previous schedules are not just discarded. We need a \textit{dynamic} algorithm!
Recomputing MVM is Inefficient!
Task Migration in the Graph

**Intuition**

For some task that just arrived, any reachable task can be preempted.
Task Migration in the Graph

**Intuition**

We just need to shift tasks by taking the *complementary* edges in the path.
Updating the Matching

**Intuition**

1) Task arrives

2) Preempt the lowest-priority reachable task
Shifting Tasks with Graph Search
Shifting Tasks with Graph Search
Shifting Tasks with Graph Search
Preempting the lowest-priority task produces an MVM!

Migrations determined via backtracking
Shifting Tasks with Graph Search

Scheduling decisions updated dynamically via BFS (linear in the size of the graph).

Migrations determined via backtracking

Preempting the lowest-priority task produces an MVM!
This Talk

Limitations of current APA schedulers

How to perform task shifting

Schedulability Analysis

Evaluation
Analyzing Strong APA Scheduling

- Previous work: *Schedulability analysis for APA scheduling* [1]
  - Works only with Linux’s migration semantics

- Recently: *Linear-programming-based response-time analysis* [2]
  - Faster in practice

We extend the LP-based RTA to consider task shifting!

Shifting Reduces Task Interference

Affinities

Weak APA (Linux)
Shifting Reduces Task Interference

Interference (due to task 1 executing)

Affinities

\( T_1 \) \( T_2 \) \( T_3 \)

Weak APA (Linux)
Shifting Reduces Task Interference

Interference (due to task 1 executing)

Affinities

- $T_1$
- $T_2$
- $T_3$

Weak APA (Linux)

Interference (due to task 1 not shifting)

Affinities

- $T_1$
- $T_2$
- $T_3$

Strong APA
The interference incurred by $T_3$ is bounded by the time that high priority tasks cannot shift outside $T_3$’s affinity.

This bound is valid only for a single migration!
Accounting for K-hop Shifting

$T_0$

Analyzed Task
Accounting for **K-hop** Shifting

**Analyzed Task** $T_0$

**Processor** $P_0$

**Processor** $P_{k-1}$

**Processor** $P_k$

Shifting $k$ tasks
Accounting for K-hop Shifting

Interference induced by $T_k$ on $T_0$ is bounded by workload of high-priority tasks on $P_k$

Details in the paper!
This Talk

- Limitations of current APA schedulers
- How to perform task shifting
- Schedulability Analysis
- Evaluation
Two Questions about Strong APA Scheduling

• To which extent does enabling task shifting prevent deadline misses?
• Assuming non-zero migration overheads, do the additional task migrations penalize the benefits of shifting?
Phase 1: Task Set Generation

1) For each point, 800 randomly generated task sets (Emberson et al.’s method [1])

2) Fixed-Priority tasks: DkC order [2]

3) Random generation of affinity assignments
   - Try to emulate application requirements
   - More details in the paper

Phase 2: Schedulability Tests

Weak APA

Sim-Weak: Simulation of APA scheduling without shifting
RTA-Weak: Previous response-time analysis for Linux

Strong APA

Sim-Strong: Simulation of APA scheduling with shifting
RTA-Strong: New LP-based response-time analysis
Analysis vs. Simulation

Simulation
- Upper Bound
  - Failure $\Rightarrow$ not schedulable
    (necessary condition)

Analysis
- Lower Bound
  - Success $\Rightarrow$ schedulable
    (sufficient condition)
Question 1

• To which extent does enabling task shifting prevent deadline misses?
Schedulability Curve

Higher is better

fraction of schedulable task sets

utilization of the task set
Benefits of Task Shifting
(8 CPUs, 12 tasks)

fraction of schedulable task sets

utilization of the task set

RTA-Weak
RTA-Strong
SIM-Weak
SIM-Strong

Strong APA
Weak APA
Benefits of Task Shifting
(8 CPUs, 12 tasks)

Strong APA improves schedulability!
Benefits of Task Shifting
(8 CPUs, 12 tasks)

Strong APA improves schedulability!

Lower bound
Strong APA ≥
Upper bound
Weak APA
Question 2

- Assuming non-zero migration overheads, do the additional task migrations penalize the benefits of shifting?
Effect of Migration Overheads

(4 CPUs, 7 tasks)
Pessimism in Overhead Analysis

Conservative results:

Analysis assumes statically that all arrivals and completions cause every task to shift.
Pessimism in Overhead Analysis

Conservative results:

Analysis assumes statically that all arrivals and completions cause every task to shift.

Tighter bounds on the number of shifts depend on task arrival patterns!
Conclusion

• We proposed new migration semantics called **strong APA scheduling**, with better temporal guarantees and maintaining API compatibility with major OSs.

• We presented a **dynamic algorithm** for scheduling decisions based on task shifting.

• Strong APA scheduling **significantly improves schedulability** (assuming negligible overheads). Migration overheads can still be analyzed (with pessimism).