The BEST Desktop Soft Real-Time Scheduler

Scott Banachowski and Scott Brandt
University of California, Santa Cruz
sbanacho@cs.ucsc.edu
Background/Motivation

• Best-effort Scheduling
  – PROS: Simple, easy, flexible, no \textit{a priori} information
  – CONS: No service guarantees, admission control = missed deadlines

• Soft Real-time Scheduling (SRT)
  – PROS: Soft guarantees, deadline-aware scheduling
  – CONS: Programming overhead, SRT API, \textit{a priori} information, inflexible

• \textbf{Best-effort scheduler Enhanced for Soft real-Time}
  – Auto-detect SRT applications/periods, boost SRT priorities, schedule accordingly
  – Result: Good SRT performance with Best-Effort programming and runtime model
Outline  **Best-Effort Enhanced for Soft-real Time**

- Related Work
- BEST Scheduler Details
- Experimental Results
- Conclusion: By auto-detecting periodic deadlines of applications, the BEST scheduler boosts performance of soft real-time applications.
Soft Real-Time CPU Scheduling

Related Work

• **Real-Time** [Liu73,Burns91…]
  – Advantage: Classical results, proven to work
  – Disadvantage: Must characterize the workload prior/inflexible

• **Proportional (Rate) Scheduling**
  [Waldspurger95,Stoica96,Nieh97,Bavier98,Duda99…]
  – Advantage: Admission control, rate and QoS guarantees
  – Disadvantage: Need APIs or interface to scheduler

• **Hierarchical Soft Real-Time Schedulers**
  [Ford96,Yodaikin97,Candea98,Regehr01…]
  – Advantage: Arbitrary scheduling arrangements
  – Disadvantage: Must “architect” the system

• **SRT solutions do not use the best-effort model, i.e. they are not good for general purpose users.**
Single-threaded Multimedia SRT Applications

• *mpeg_play* video player
  – processes and displays one frame per period
  – uses OS timer to wake up for next frame
  – wakes up twice per frame, once for frame period and once for video synchronization
  – measured period is twice the frame period (“dual-period effect”)

• *mpg123* audio player
  – fills the sound buffer during each period
  – sleeps for a nominal period between periods (no synchronization)
  – measured period is sleep time + processing time
BEST Scheduler Design Goals

1. Same scheduling policy for all applications
2. No \textit{a priori} information of resource needs
3. Performance enhancement for SRT
   - Processes with predictable periodic behavior should receive priority boots
   - Preemptive scheduler based on classical algorithm
4. Reasonable/consistent default behavior
   - Favor I/O bound over CPU bound processes
   - No starvation and graceful degradation when overloaded
Linux Scheduler

- Scheduler chooses the process with the highest dynamic priority from *runnable queue*
  - scheduler executes after servicing interrupts
  - `wake_up_process()` is the entry point into the runnable queue

- Simulates a multi-level feedback queue
  - The dynamic priority is the time quantum, and decrements as a process executes
  - If all runnable jobs have zero priority, all applications receive priority increase
BEST Scheduler

1. Monitor ready queue entry times to detect SRT application periods and infer deadlines
2. Filter out non-periodic processes with confidence measure
3. Assign pseudo-deadlines to non-periodic processes to get “normal” best-effort behavior
4. Schedule all applications with Earliest Deadline First
BEST Implementation Details

- **In** `wake_up_process()`
  - measure time elapsed since the previous wake up ($P_{est}$)
  - calculate a weighted period $P_n = (P_{est} + w \cdot P_{n-1}) / (1+w)$
  - calculate confidence (an estimate of variance)
  \[ |P_{est} \mod P_n - P_n/2| \]
  - if (confidence>threshold) next$_p$ _deadline= current_time + $P_n$
  - else next$_p$ _deadline = current_time + $P_{max}$
  - set interval timer (max proportion of CPU allowed per period)

- **In** `schedule()`
  - If current process interval timer expired, set
  next$_p$ _deadline= current_time + $P_{max} + offset$
  - Select process with the earliest deadline
• If a process misses a deadline, it still has an opportunity for high confidence
• A process with a “dual-period effect” is still detectable
loop & periodic (0.1s 40%)

Linux

BEST

No missed deadlines
loop & periodic (0.1s 70%)

Linux

BEST

RM

Missed 29% of its deadlines

No missed deadlines
loop & periodic (1s 30%) (0.1s 30%)

Linux

BEST

Missed 0% and 6% of deadlines

No missed deadlines
loop & periodic (1s 30%) (0.5s 30%) (0.1s 30%)

Linux

BEST

RM

Missed 83%, 80% and 10% of deadlines

No missed deadlines
loop & periodic (0.61s 31%) (0.43 30%) (0.13 31%)

BEST

No missed deadlines

RM

The long period application missed 27% of its deadlines
loop & periodic (1s 40%) (0.5s 40%) (0.1s 40%)

Linux

BEST

RM

Missed 98%, 93% and 19% of deadlines

Missed 71%, 5%, and 2% of deadlines

Missed 98%, 0%, and 0% of deadlines
Conclusion

• It is relatively easy to detect SRT application periods
• Supporting multimedia and soft real-time using the Best-Effort model is feasible when you can detect periodic deadlines
• BEST meets its design goal by
  – Auto-detecting periodic behavior and boosting SRT performance
  – Retaining BE programming and runtime model
• Future work: Tune scheduler using real application workloads