SMART: A CPU Scheduler for Multimedia Applications

Application Requirements

- Application demands are changing rapidly
- Past: textual applications
  - Require only a best-effort service
  - Easy to schedule
- Present and future: multimedia applications
  - Have strict latency requirements
  - Difficult to schedule an application mix

Scheduler Demands

- Soft real-time constraints
  - Data useful only if received on time
  - Missed deadlines are not catastrophic
- Insatiable resource demands
  - Must handle overload conditions
- Dynamically adaptive applications
- Co-existence of continuous media and conventional applications

SMART: Principles

- Priority: Lower priority task should not affect a higher priority task
- Proportional sharing
- Handle load fluctuations
- Provide quality of service: meet deadlines, low latencies
- Trade instantaneous fairness for better quality
- Notification of resource availability

SMART Operation

- Process schedule $= f(\text{importance}, \text{urgency})$
- Importance: Use priorities and weighted fair queuing
- Urgency: Use EDF scheduling

SMART Scheduling Principles

- Each task is assigned a tuple (P, S)
  - P : priority assigned to a process
  - S : share of cpu bandwidth within a priority level
- Active tasks progress at the rate of $1/S$
- Similar to WFQ

SMART Scheduling Principles (contd)

- Virtual finishing time (VFT): virtual time + Q/S
- Use bias to compute VFT
  - Measure of delay tolerance of tasks
  - Real-time tasks have zero bias
  - Reset to 0 on user interaction
  - Special handling for tasks using less than their fair share
- Biased VFT (BVFT): VFT + b/S

SMART Scheduling Algorithm

- Maintain a tuple $(P, BVFT)$ per task
- $(P_1, BVFT_1) > (P_2, BVFT_2)$ if
  - $P_1 > P_2$ or
  - $P_1 = P_2$ and $BVFT_1 < BVFT_2$
- **Algorithm**
  - If largest tuple is a best-effort task, schedule it
  - Else choose all real-time tasks with tuples larger than the largest best-effort task
    - Use best-effort real-time scheduling to order these tasks
    - Choose task with the largest tuple value and insert it into a feasible EDF schedule
    - Schedule task with earliest deadline from this schedule

Complexity

- Managing the tuple list: $O(N)$
  - Can be reduced to $O(logN)$ by using a tree data structure
- Creating the working schedule: $O(N_R^2)$
  - Can be reduced to $O(N_R)$ using optimizations

SMART Interface

- Application developer interface
  - Time constraints: application provides time bounds
  - Notifications: scheduler provides notification of constraint failure
- User interface
  - Priorities: strict priority scheduling
  - Weights: distribute CPU bandwidth in proportion to weights within a priority level

Performance Results

- Application mix
  - Batch: Dhrystone (CPU intensive)
  - Interactive: Typing (I/O intensive)
  - Soft real-time: media player
- SMART performs better than UNIX TS, SVR4-RT, and WFQ