Scheduling Tasks with Mixed Preemption Relations for Robustness to Timing Faults

John Regehr
University of Utah
12/5/2002

Preview

- Scheduling Tasks…
  - With static priorities
- With Mixed Preemption Relations…
  - Two new abstractions
  - + algorithms
- For Robustness to Timing Faults
  - A new optimality criterion
  - + algorithms

Review: Preemption Threshold Scheduling

- Saksena and Wang (RTSS 2000)
- Task instance has two priorities:
  - Normal – release to first execution
  - Preemption threshold – first execution to completion of instance
- Subsumes pure preemptive and pure non-preemptive scheduling
  - Also, significant schedulability gains in practice

More Review

- Key insight: Mutually non-preemptible tasks can run in the same thread
  - Fewer stacks → Save memory
  - Fewer context switches → Save CPU time
- Scheduling algorithm can minimize threads
  - But this is a back-end optimization
- This talk: making mixed preemption first-class

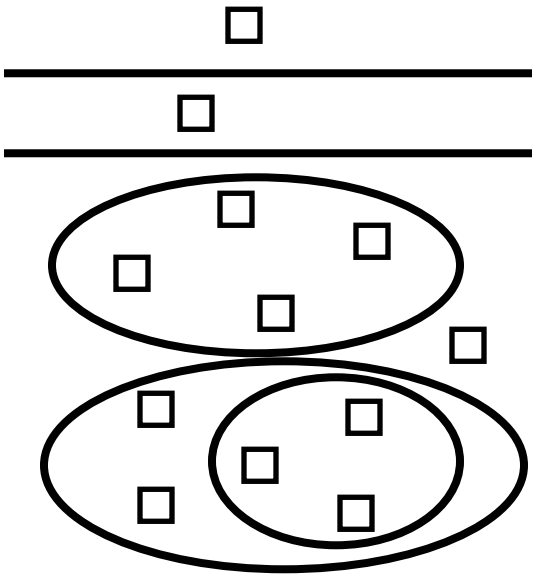
Task Clusters

- Collection of tasks that must be mutually non-preemptible
- Why?
  - Developing concurrent code is a serious problem in practice
    - Races and deadlocks are easy to avoid when writing non-preemptive code
  - Sharing is more efficient
    - No locks needed for resources with cluster scope

Task Barriers

- Partitions tasks into subsets that must be mutually preemptible
- Why?
  - Respect architectural constraints: interrupt handlers, bottom-half handlers, etc.
  - Avoid undesirable coupling between tasks

Clusters and Barriers



Scheduling Task Clusters and Barriers

- Strategy
  - Minimize threads
  - Respect constraints imposed by clusters and barriers
  - Use a heuristic search – optimal algorithms are exponential
- Can target:
  - Systems supporting preemption thresholds
  - Standard RTOSs

But we're not quite done…

1. **Question: Should we always choose the valid schedule that minimizes threads?**
2. **Generalized question: When there are multiple feasible schedules for a system, how to choose one?**
   - **Traditional answer: just take the first feasible schedule found**
   - **Not good enough!**

Always Minimize Threads?

- **Answer 1: If WCET is really worst-case then YES**
- **Answer 2: If there is uncertainty about "WCET" then NO**
  - **Reducing threads can increase missed deadlines during timing faults**
- **Timing fault: task instance that overruns but returns correct result**

When is "WCET" not WC?

- When WCET is too pessimistic
- When using probabilistic WCET (York – RTSS 2001 & 2002)
- When no WCET tool is used
  - WCET ≠ WMET + FF

But we're not quite done…

1. Question: Should we always choose the valid schedule that minimizes threads?
2. Generalized question: When there are multiple feasible schedules for a system, how to choose one?
   - Traditional answer: just take the first feasible schedule found
   - Not good enough!

How to Pick a Schedule?

- Pick a robust schedule
  - Minimizes missed deadlines under timing faults
  - Need a model for timing faults
- Uniform expansion in run-time
  - Lehoczky et al. (1989) used this as a measure of schedulability
  - Critical scaling factor or Δ^*: largest expansion that leaves a feasible task set

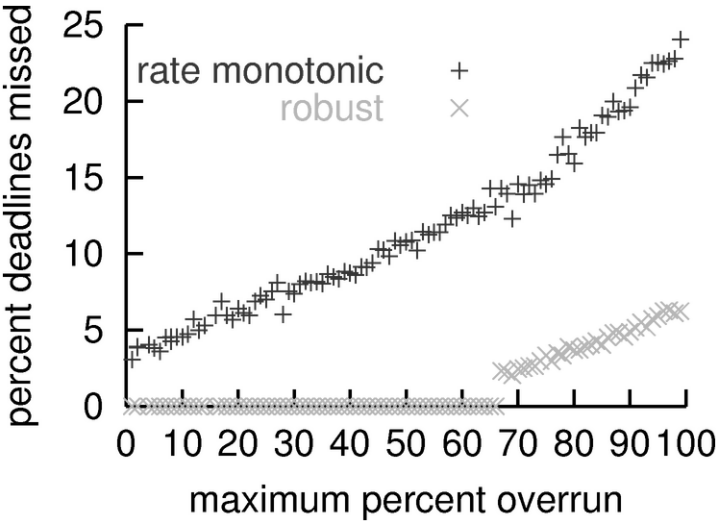
Robust Scheduling

- For a given class of task sets, a robust-optimal algorithm finds a feasible schedule with the maximal critical scaling factor
- Claim: robust schedules are better

Justification for Claim

- Consider:
  - t_0 : C = 400; T,D = 1999, J = 0
  - t_1 : C = 400; T,D = 2000, J = 1200
- RM or DM schedule:
  - t_0 higher priority than t_1
  - Δ^* approx 1.00
- Robust schedule:
  - t_1 higher priority than t_0
  - Δ^* approx 1.67

Justification Cont'd



Basic Result

- RM and DM are robust-optimal for the class of task sets where they are traditionally optimal
  - No jitter
- Why?
  - Neither RM or DM takes WCET into account when creating a schedule

More Basic Results

- The following algorithms are not robust-optimal:
  - Audsley's optimal algorithms for preemptive and non-preemptive scheduling
  - All existing algorithms for scheduling tasks with preemption thresholds
- Proof is by counterexample

Finding Robust Schedules

- Binary search
  - When schedulability is monotonic in the scaling factor AND
  - An optimal algorithm for finding a feasible schedule is available
- Heuristic search
  - When above conditions are not met
  - Works well in practice
- See paper for evaluation…

Toward A Software Process

1. Express system design in terms of task clusters and barriers
2. Jointly
   - Minimize threads
   - Maximize critical scaling factor
3. Now the system architect can make an informed tradeoff

Avionics Example

- Task set from Tindell et al. (RTSJ March 1994)
  - 17 tasks + clock interrupt
- Modeling the system:
  - Use barrier to isolate interrupt
  - Put synchronizing tasks in clusters
- Nice result: no locks needed at run time

Avionics Cont'd

- Best schedule with preemption thresholds:
  - 5 threads, $\Delta^* = 1.265$
- Best schedules with static priorities:
  - 5 threads, $\Delta^* = 1.104$
  - 6 threads, $\Delta^* = 1.174$

Future Work

- More abstractions
  - Reservation-based scheduling with task clusters?
- Approximate scheduling algorithms
- Explore different models for timing faults

Conclusions

- Task clusters and barriers
  - Better abstractions to protect developers from multithreading
- Robust scheduling
  - A new optimality criterion
  - Useful when there is uncertainty about run-time
  - Broadly applicable
    - No need to buy into task clusters and barriers

The End

- Code for all new algorithms is available:
  - http://www.cs.utah.edu/~regehr/spak